IBM DB2 10.1
for Linux, UNIX, and Windows

SQL Reference Volume 2

Updated January, 2013

SC27-3886-01

IBM DB2 10.1
for Linux, UNIX, and Windows

SQL Reference Volume 2

Updated January, 2013

SC27-3886-01

Note
Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 1357.

Edition Notice

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at http://www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at http://www.ibm.com/planetwide/

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this book vii
Who should use this book vii
How this book is structured vii
How to read the syntax diagrams viii
Conventions used in this manual x
Error conditions x
Highlighting conventions x
Related documentation x

SQL statements 1
How SQL statements are invoked 10
Embedding a statement in an application program 10
Dynamic preparation and execution 11
Static invocation of a select-statement 11
Dynamic invocation of a select-statement . . . 12
Interactive invocation 12
SQL use with other host systems 12
Detecting and processing error and warning conditions in host language applications 12
SQL comments 13
Conditional compilation in SQL 14
About SQL control statements 17
References to SQL parameters, SQL variables, and global variables 17
References to SQL labels 18
References to SQL condition names 18
References to SQL statement names 18
References to SQL cursor names 19
Function, method, and procedure designators . . . 20
ALLOCATE CURSOR 24
ALTER AUDIT POLICY 26
ALTER BUFFERPOOL 29
ALTER DATABASE PARTITION GROUP 32
ALTER DATABASE 36
ALTER EVENT MONITOR 41
ALTER FUNCTION 46
ALTER HISTOGRAM TEMPLATE 50
ALTER INDEX 52
ALTER MASK 53
ALTER METHOD 54
ALTER MODULE 56
ALTER NICKNAME 64
ALTER PACKAGE 73
ALTER PERMISSION 76
ALTER PROCEDURE (external) 77
ALTER PROCEDURE (sourced) 80
ALTER PROCEDURE (SQL) 82
ALTER SCHEMA 84
ALTER SECURITY LABEL COMPONENT 86
ALTER SECURITY POLICY 89
ALTER SEQUENCE 93
ALTER SERVER 97
ALTER SERVICE CLASS 100
ALTER STOGROUP 110
ALTER TABLE 114
ALTER TABLESPACE 179
ALTER THRESHOLD 194
ALTER TRIGGER 207
ALTER TRUSTED CONTEXT 208
ALTER TYPE (structured) 216
ALTER USAGE LIST 223
ALTER USER MAPPING 225
ALTER VIEW 227
ALTER WORK ACTION SET 229
ALTER WORK CLASS SET 243
ALTER WORKLOAD 249
ALTER WRAPPER 265
ALTER XSROBJECT 267
ASSOCIATE LOCATORS 268
AUDIT 270
BEGIN DECLARE SECTION 274
CALL 276
CASE 284
CLOSE 287
COMMENT 289
COMMIT 300
Compound SQL 302
Compound SQL (inlined) 303
Compound SQL (embedded) 308
Compound SQL (compiled) 312
CONNECT (type 1) 329
CONNECT (type 2) 336
CREATE ALIAS 343
CREATE AUDIT POLICY 347
CREATE BUFFERPOOL 350
CREATE DATABASE PARTITION GROUP . . . 354
CREATE EVENT MONITOR 356
CREATE EVENT MONITOR (activities) 376
CREATE EVENT MONITOR (change history) . . 387
CREATE EVENT MONITOR (locking) 394
CREATE EVENT MONITOR (package cache) statement 400
CREATE EVENT MONITOR (statistics) 407
CREATE EVENT MONITOR (threshold violations) 419
CREATE EVENT MONITOR (unit of work) . . . 430
CREATE FUNCTION 435
CREATE FUNCTION (external scalar) 436
CREATE FUNCTION (external table) 464
CREATE FUNCTION (OLE DB external table) . . 484
CREATE FUNCTION (sourced or template) . . . 495
CREATE FUNCTION (SQL scalar, table, or row) 509
CREATE FUNCTION MAPPING 526
CREATE GLOBAL TEMPORARY TABLE 530
CREATE HISTOGRAM TEMPLATE 543
CREATE INDEX 545
CREATE INDEX EXTENSION 566
CREATE MASK 572
CREATE METHOD 578
CREATE MODULE 584
CREATE NICKNAME 586

CREATE PERMISSION 599
CREATE PROCEDURE 603
CREATE PROCEDURE (external) 604
CREATE PROCEDURE (sourced) 620
CREATE PROCEDURE (SQL) 626
CREATE ROLE 636
CREATE SCHEMA 637
CREATE SECURITY LABEL COMPONENT . . . 640
CREATE SECURITY LABEL 643
CREATE SECURITY POLICY 645
CREATE SEQUENCE 647
CREATE SERVICE CLASS 653
CREATE SERVER 664
CREATE STOGROUP 668
CREATE SYNONYM 671
CREATE TABLE 672
CREATE TABLESPACE 752
CREATE THRESHOLD 767
CREATE TRANSFORM 784
CREATE TRIGGER 788
CREATE TRUSTED CONTEXT 803
CREATE TYPE 810
CREATE TYPE (array) 811
CREATE TYPE (cursor) 817
CREATE TYPE (distinct) 820
CREATE TYPE (row) 828
CREATE TYPE (structured) 833
CREATE TYPE MAPPING 856
CREATE USAGE LIST 863
CREATE USER MAPPING 867
CREATE VARIABLE 869
CREATE VIEW 879
CREATE WORK ACTION SET 894
CREATE WORK CLASS SET 903
CREATE WORKLOAD 908
CREATE WRAPPER 926
DECLARE CURSOR 928
DECLARE GLOBAL TEMPORARY TABLE . . . 934
DELETE 947
DESCRIBE 957
DESCRIBE INPUT 958
DESCRIBE OUTPUT 962
DISCONNECT 966
DROP 969
END DECLARE SECTION 1002
EXECUTE 1003
EXECUTE IMMEDIATE 1011
EXPLAIN 1014
FETCH 1019
FLUSH BUFFERPOOLS 1023
FLUSH EVENT MONITOR 1024
FLUSH FEDERATED CACHE 1025
FLUSH OPTIMIZATION PROFILE CACHE . . . 1027
FLUSH PACKAGE CACHE 1029
FOR 1030
FREE LOCATOR 1033
GET DIAGNOSTICS 1034
GOTO 1037
GRANT (database authorities) 1039
GRANT (exemption) 1044
GRANT (global variable privileges) 1047
GRANT (index privileges) 1049
GRANT (module privileges) 1051
GRANT (package privileges) 1053
GRANT (role) 1056
GRANT (routine privileges) 1059
GRANT (schema privileges) 1063
GRANT (security label) 1066
GRANT (sequence privileges) 1069
GRANT (server privileges) 1072
GRANT (SETSESSIONUSER privilege) 1074
GRANT (table space privileges) 1076
GRANT (table, view, or nickname privileges) . . 1078
GRANT (workload privileges) 1084
GRANT (XSR object privileges) 1086
IF 1087
INCLUDE 1089
INSERT 1091
ITERATE 1102
LEAVE 1104
LOCK TABLE 1106
LOOP 1108
MERGE 1110
OPEN 1122
PIPE 1128
PREPARE 1130
REFRESH TABLE 1136
RELEASE (connection) 1140
RELEASE SAVEPOINT 1142
RENAME 1143
RENAME STOGROUP 1145
RENAME TABLESPACE 1146
REPEAT 1147
RESIGNAL 1149
RETURN 1152
REVOKE (database authorities) 1154
REVOKE (exemption) 1158
REVOKE (global variable privileges) 1160
REVOKE (index privileges) 1162
REVOKE (module privileges) 1164
REVOKE (package privileges) 1166
REVOKE (role) 1169
REVOKE (routine privileges) 1171
REVOKE (schema privileges) 1175
REVOKE (security label) 1177
REVOKE (sequence privileges) 1179
REVOKE (server privileges) 1181
REVOKE (SETSESSIONUSER privilege) 1183
REVOKE (table space privileges) 1185
REVOKE (table, view, or nickname privileges) 1187
REVOKE (workload privileges) 1192
REVOKE (XSR object privileges) 1194
ROLLBACK 1195
SAVEPOINT 1198
SELECT 1201
SELECT INTO 1202
SET COMPILATION ENVIRONMENT 1206
SET CONNECTION 1207
SET CURRENT DECFLOAT ROUNDING MODE 1209
SET CURRENT DEFAULT TRANSFORM GROUP 1211
SET CURRENT DEGREE 1212
SET CURRENT EXPLAIN MODE 1214

SET CURRENT EXPLAIN SNAPSHOT 1217
SET CURRENT FEDERATED ASYNCHRONY 1219
SET CURRENT IMPLICIT XMLPARSE OPTION 1221
SET CURRENT ISOLATION 1222
SET CURRENT LOCALE LC_MESSAGES . . . 1223
SET CURRENT LOCALE LC_TIME 1225
SET CURRENT LOCK TIMEOUT 1227
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 1229
SET CURRENT MDC ROLLOUT MODE 1231
SET CURRENT OPTIMIZATION PROFILE . . . 1233
SET CURRENT PACKAGE PATH 1236
SET CURRENT PACKAGESET 1240
SET CURRENT QUERY OPTIMIZATION . . . 1242
SET CURRENT REFRESH AGE 1245
SET CURRENT SQL_CCFLAGS 1247
SET CURRENT TEMPORAL BUSINESS_TIME 1249
SET CURRENT TEMPORAL SYSTEM_TIME . . 1251
SET ENCRYPTION PASSWORD 1253
SET EVENT MONITOR STATE 1255
SET INTEGRITY 1257
SET PASSTHRU 1276
SET PATH 1278
SET ROLE 1280
SET SCHEMA 1281
SET SERVER OPTION 1283
SET SESSION AUTHORIZATION 1285
SET USAGE LIST STATE 1288
SET variable 1291
SIGNAL 1303
TRANSFER OWNERSHIP 1306
TRUNCATE 1318
UPDATE 1321
VALUES 1337
VALUES INTO 1338
WHENEVER 1341
WHILE 1344

Appendix A. Overview of the DB2 technical information 1347
DB2 technical library in hardcopy or PDF format 1348
Displaying SQL state help from the command line processor 1350
Accessing different versions of the DB2 Information Center 1350
Updating the DB2 Information Center installed on your computer or intranet server 1351
Manually updating the DB2 Information Center installed on your computer or intranet server . . 1352
DB2 tutorials 1354
DB2 troubleshooting information 1354
Terms and conditions 1355

Appendix B. Notices 1357

Index 1361

About this book

The SQL Reference in its two volumes defines the SQL language used by DB2® Database for Linux, UNIX, and Windows.

It includes:

- Information about relational database concepts, language elements, functions, and the forms of queries (Volume 1)
- Information about the syntax and semantics of SQL statements (Volume 2)

Who should use this book

This book is intended for anyone who wants to use the Structured Query Language (SQL) to access a database. It is primarily for programmers and database administrators, but it can also be used by those who access databases through the command line processor (CLP).

This book is a reference rather than a tutorial. It assumes that you will be writing application programs and therefore presents the full functions of the database manager.

How this book is structured

The second volume of the *SQL Reference* contains information about the syntax and semantics of SQL statements.

- "Statements" contains syntax diagrams, semantic descriptions, rules, and examples of all SQL statements, including SQL procedure statements.

How to read the syntax diagrams

This topic describes the structure of SQL syntax diagrams.

Read the syntax diagrams from left to right and top to bottom, following the path of the line.

The ▸▸— symbol indicates the beginning of a syntax diagram.

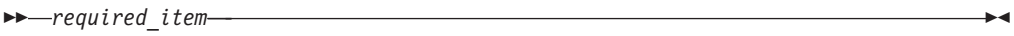
The —▸ symbol indicates that the syntax is continued on the next line.

The ▸— symbol indicates that the syntax is continued from the previous line.

The —▸◂ symbol indicates the end of a syntax diagram.

Syntax fragments start with the ├— symbol and end with the —┤ symbol.

Required items appear on the horizontal line (the main path).

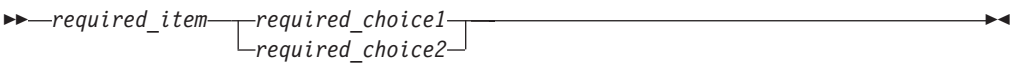


Optional items appear below the main path.

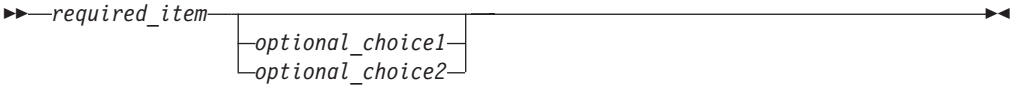
If an optional item appears above the main path, that item has no effect on execution, and is used only for readability.

If you can choose from two or more items, they appear in a stack.

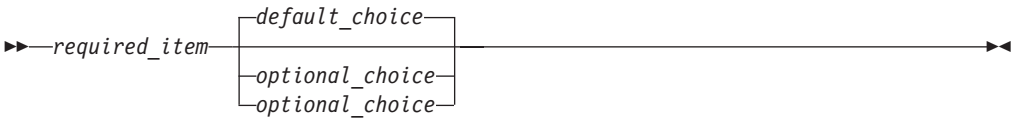
If you *must* choose one of the items, one item of the stack appears on the main path.



If choosing one of the items is optional, the entire stack appears below the main path.

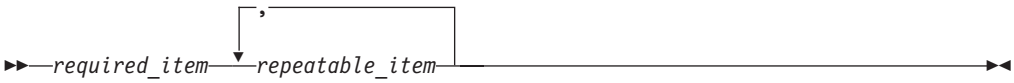


If one of the items is the default, it will appear above the main path, and the remaining choices will be shown below.



An arrow returning to the left, above the main line, indicates an item that can be repeated. In this case, repeated items must be separated by one or more blanks.

If the repeat arrow contains a comma, you must separate repeated items with a comma.

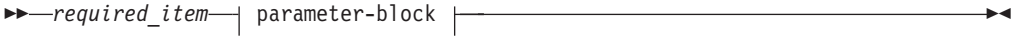


A repeat arrow above a stack indicates that you can make more than one choice from the stacked items or repeat a single choice.

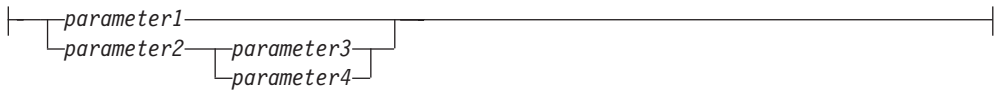
Keywords appear in uppercase (for example, `FROM`). They must be spelled exactly as shown. Variables appear in lowercase (for example, `column-name`). They represent user-supplied names or values in the syntax.

If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sometimes a single variable represents a larger fragment of the syntax. For example, in the following diagram, the variable `parameter-block` represents the whole syntax fragment that is labeled **parameter-block**:



parameter-block:



Adjacent segments occurring between "large bullets" (●) may be specified in any sequence.

The above diagram shows that item2 and item3 may be specified in either order. Both of the following are valid:

```
required_item item1 item2 item3 item4
required_item item1 item3 item2 item4
```

Conventions used in this manual

Error conditions

An error condition is indicated within the text of the manual by listing the SQLSTATE associated with the error in parentheses.

For example:

```
A duplicate signature returns an SQL error (SQLSTATE 42723).
```

Highlighting conventions

This topic covers the conventions used in the SQL Reference.

- **Bold** indicates commands, keywords, and other items whose names are predefined by the system.
- *Italics* indicates one of the following items:
  - Names or values (variables) that must be supplied by the user
  - General emphasis
  - The introduction of a new term
  - A reference to another source of information

Related documentation

The following publications might prove useful when you are preparing applications:

- *Getting Started with Database Application Development*
  - Provides an introduction to DB2 application development, including platform prerequisites; supported development software; and guidance on the benefits and limitations of the supported programming APIs.
- *DB2 for i5/OS SQL Reference*
  - This book defines SQL as supported by DB2 Query Manager and SQL Development Kit on System i®. It contains reference information for the tasks of system administration, database administration, application programming, and operation. This manual includes syntax, usage notes, keywords, and examples for each of the SQL statements used on i5/OS® systems running DB2.
- *DB2 for z/OS SQL Reference*
  - This book defines SQL used in DB2 for z/OS®. It provides query forms, SQL statements, SQL procedure statements, DB2 limits, SQLCA, SQLDA, catalog tables, and SQL reserved words for z/OS systems running DB2.
- *DB2 Spatial Extender User's Guide and Reference*
  - This book discusses how to write applications to create and use a geographic information system (GIS). Creating and using a GIS involves supplying a database with resources and then querying the data to obtain information such as locations, distances, and distributions within areas.
- *IBM SQL Reference*
  - This book contains all the common elements of SQL that span IBM's database products. It provides limits and rules that assist in preparing portable programs using IBM databases. This manual provides a list of SQL extensions and incompatibilities among the following standards and products: SQL92E, XPG4-SQL, IBM-SQL, and the IBM relational database products.
- *American National Standard X3.135-1992, Database Language SQL*

  - Contains the ANSI standard definition of SQL.
- *ISO/IEC 9075:1992, Database Language SQL*
  - Contains the 1992 ISO standard definition of SQL.
- *ISO/IEC 9075-2:2003, Information technology -- Database Languages -- SQL -- Part 2: Foundation (SQL/Foundation)*
  - Contains a large portion of the 2003 ISO standard definition of SQL.
- *ISO/IEC 9075-4:2003, Information technology -- Database Languages -- SQL -- Part 4: Persistent Stored Modules (SQL/PSM)*
  - Contains the 2003 ISO standard definition for SQL procedure control statements.

Related documentation

SQL statements

This topic contains tables that list the SQL statements classified by type.

- SQL schema statements (Table 1)
- SQL data change statements (Table 2 on page 6)
- SQL data statements (Table 3 on page 6)
- SQL transaction statements (Table 4 on page 7)
- SQL connection statements (Table 5 on page 7)
- SQL dynamic statements (Table 6 on page 7)
- SQL session statements (Table 7 on page 7)
- SQL embedded host language statements (Table 8 on page 9)
- SQL control statements (Table 9 on page 9)

Table 1. SQL schema statements

SQL Statement	Purpose
"ALTER AUDIT POLICY" on page 26	Modifies the definition of an audit policy at the current server.
"ALTER BUFFERPOOL" on page 29	Changes the definition of a buffer pool.
"ALTER DATABASE" on page 36	Adds new storage paths to the collection of paths that are used for automatic storage table spaces.
"ALTER EVENT MONITOR" on page 41	Changes the definition of a TABLE or UNFORMATTED EVENT TABLE event monitor.
"ALTER DATABASE PARTITION GROUP" on page 32	Changes the definition of a database partition group.
"ALTER FUNCTION" on page 46	Modifies an existing function by changing the properties of the function.
"ALTER HISTOGRAM TEMPLATE" on page 50	Modifies the template describing the type of histogram that can be used to override one or more of the default histograms of a service class or a work class.
"ALTER INDEX" on page 52	Changes the definition of an index.
"ALTER MASK" on page 53	Changes the definition of a column mask.
"ALTER METHOD" on page 54	Modifies an existing method by changing the method body associated with the method.
"ALTER MODULE" on page 56	Changes the definition of a module.
"ALTER NICKNAME" on page 64	Changes the definition of a nickname.
"ALTER PACKAGE" on page 73	Alters bind options for a package at the current server without having to bind or rebind the package.
"ALTER PERMISSION" on page 76	Changes the definition of a row permission.
"ALTER PROCEDURE (external)" on page 77	Modifies an existing external procedure by changing the properties of the procedure.
"ALTER PROCEDURE (sourced)" on page 80	Modifies an existing sourced procedure by changing the data type of one or more parameters of the sourced procedure.
"ALTER PROCEDURE (SQL)" on page 82	Modifies an existing SQL procedure by changing the properties of the procedure.
"ALTER SCHEMA" on page 84	Modifies an existing schema by changing the data capture attribute of the schema.

Table 1. SQL schema statements (continued)

SQL Statement	Purpose
"ALTER SECURITY LABEL COMPONENT" on page 86	Modifies a security label component.
"ALTER SECURITY POLICY" on page 89	Modifies a security policy.
"ALTER SEQUENCE" on page 93	Changes the definition of a sequence.
"ALTER SERVER" on page 97	Changes the definition of a data source in a federated system.
"ALTER SERVICE CLASS" on page 100	Changes the definition of a service class.
"ALTER STOGROUP" on page 110	Changes the definition of a storage group.
"ALTER TABLE" on page 114	Changes the definition of a table.
"ALTER TABLESPACE" on page 179	Changes the definition of a table space.
"ALTER THRESHOLD" on page 194	Changes the definition of a threshold.
"ALTER TRIGGER" on page 207	Changes the definition of a trigger.
"ALTER TRUSTED CONTEXT" on page 208	Changes the definition of a trusted context at the current server.
"ALTER TYPE (structured)" on page 216	Changes the definition of a structured type.
"ALTER USAGE LIST" on page 223	Changes the definition of a usage list.
"ALTER USER MAPPING" on page 225	Changes the definition of a user authorization mapping.
"ALTER VIEW" on page 227	Changes the definition of a view by altering a reference type column to add a scope.
"ALTER WORK ACTION SET" on page 229	Adds, alters, or drops work actions within a work action set.
"ALTER WORK CLASS SET" on page 243	Adds, alters, or drops work classes within a work class set.
"ALTER WORKLOAD" on page 249	Changes a workload.
"ALTER WRAPPER" on page 265	Updates the options that, along with a wrapper module, are used to access data sources of a specific type.
"ALTER XSROBJECT" on page 267	Enables or disables decomposition support for a specific XML schema.
"AUDIT" on page 270	Determines the audit policy that is to be used for a particular database or database object at the current server.
"COMMENT" on page 289	Replaces or adds a comment to the description of an object.
"CREATE ALIAS" on page 343	Defines an alias for a module, nickname, sequence, table, view, or another alias.
"CREATE AUDIT POLICY" on page 347	Defines an auditing policy at the current server.
"CREATE BUFFERPOOL" on page 350	Defines a new buffer pool.
"CREATE DATABASE PARTITION GROUP" on page 354	Defines a database partition group.
"CREATE EVENT MONITOR" on page 356	Specifies events in the database to monitor.
"CREATE EVENT MONITOR (activities)" on page 376	Specifies activity events in the database to monitor.
"CREATE EVENT MONITOR (change history)" on page 387	Specifies change history events in the database to monitor.
"CREATE EVENT MONITOR (locking)" on page 394	Specifies locking events in the database to monitor.
"CREATE EVENT MONITOR (package cache) statement" on page 400	Specifies package cache statement events in the database to monitor.
"CREATE EVENT MONITOR (statistics)" on page 407	Specifies statistics events in the database to monitor.

Table 1. SQL schema statements (continued)

SQL Statement	Purpose
"CREATE EVENT MONITOR (threshold violations)" on page 419	Specifies threshold violation events in the database to monitor.
"CREATE EVENT MONITOR (unit of work)" on page 430	Specifies unit of work events in the database to monitor.
"CREATE FUNCTION" on page 435	Registers a user-defined function.
"CREATE FUNCTION (external scalar)" on page 436	Registers a user-defined external scalar function.
"CREATE FUNCTION (external table)" on page 464	Registers a user-defined external table function.
"CREATE FUNCTION (OLE DB external table)" on page 484	Registers a user-defined OLE DB external table function.
"CREATE FUNCTION (sourced or template)" on page 495	Registers a user-defined sourced function or a function template.
"CREATE FUNCTION (SQL scalar, table, or row)" on page 509	Defines a user-defined SQL function.
"CREATE FUNCTION MAPPING" on page 526	Defines a function mapping.
"CREATE GLOBAL TEMPORARY TABLE" on page 530	Defines a created temporary table.
"CREATE HISTOGRAM TEMPLATE" on page 543	Defines a template describing the type of histogram that can be used to override one or more of the default histograms of a service class or a work class.
"CREATE INDEX" on page 545	Defines an index on a table.
"CREATE INDEX EXTENSION" on page 566	Defines an extension object for use with indexes on tables with structured or distinct type columns.
"CREATE MASK" on page 572	Defines a column mask.
"CREATE METHOD" on page 578	Defines a method body to associate with a previously defined method specification.
"CREATE MODULE" on page 584	Defines a module.
"CREATE NICKNAME" on page 586	Defines a nickname.
"CREATE PERMISSION" on page 599	Defines a row permission.
"CREATE PROCEDURE" on page 603	Defines a procedure.
"CREATE PROCEDURE (external)" on page 604	Defines an external procedure.
"CREATE PROCEDURE (sourced)" on page 620	Defines a procedure (the sourced procedure) that is based on another procedure (the source procedure). In a federated system, a federated procedure is a sourced procedure whose source procedure is at a supported data source.
"CREATE PROCEDURE (SQL)" on page 626	Defines an SQL procedure.
"CREATE ROLE" on page 636	Defines a role at the current server.
"CREATE SCHEMA" on page 637	Defines a schema.
"CREATE SECURITY LABEL COMPONENT" on page 640	Defines a component that is to be used as part of a security policy.
"CREATE SECURITY LABEL" on page 643	Defines a security label.
"CREATE SECURITY POLICY" on page 645	Defines a security policy.

Table 1. SQL schema statements (continued)

SQL Statement	Purpose
"CREATE SEQUENCE" on page 647	Defines a sequence.
"CREATE SERVER" on page 664	Defines a data source to a federated database.
"CREATE SERVICE CLASS" on page 653	Defines a service class.
"CREATE STOGROUP" on page 668	Defines a new storage group within the database.
"CREATE SYNONYM" on page 671	Defines a synonym for a module, nickname, sequence, table, view, or another synonym.
"CREATE TABLE" on page 672	Defines a table.
"CREATE TABLESPACE" on page 752	Defines a table space.
"CREATE THRESHOLD" on page 767	Defines a threshold.
"CREATE TRANSFORM" on page 784	Defines transformation functions.
"CREATE TRIGGER" on page 788	Defines a trigger.
"CREATE TRUSTED CONTEXT" on page 803	Defines a trusted context at the current server.
"CREATE TYPE" on page 810	Defines a user-defined data type at the current server.
"CREATE TYPE (array)" on page 811	Defines an array type.
"CREATE TYPE (cursor)" on page 817	Defines a cursor type.
"CREATE TYPE (distinct)" on page 820	Defines a distinct data type.
"CREATE TYPE (row)" on page 828	Defines a row type.
"CREATE TYPE (structured)" on page 833	Defines a structured data type.
"CREATE TYPE MAPPING" on page 856	Defines a mapping between data types.
"CREATE USAGE LIST" on page 863	Defines a usage list in order to monitor all unique sections (DML statements) that have referenced a particular table or index during their execution.
"CREATE USER MAPPING" on page 867	Defines a mapping between user authorizations.
"CREATE VARIABLE" on page 869	Defines a global variable.
"CREATE VIEW" on page 879	Defines a view of one or more table, view or nickname.
"CREATE WORK ACTION SET" on page 894	Defines a work action set and work actions within the work action set.
"CREATE WORK CLASS SET" on page 903	Defines a work class set.
"CREATE WORKLOAD" on page 908	Defines a workload.
"CREATE WRAPPER" on page 926	Registers a wrapper.
"DROP" on page 969	Deletes objects in the database.
"GRANT (database authorities)" on page 1039	Grants authorities on the entire database.
"GRANT (exemption)" on page 1044	Grants an exemption on an access rule for a specified label-based access control (LBAC) security policy.
"GRANT (global variable privileges)" on page 1047	Grants one or more privileges on a created global variable.
"GRANT (index privileges)" on page 1049	Grants the CONTROL privilege on indexes in the database.
"GRANT (module privileges)" on page 1051	Grants privileges on a module.
"GRANT (package privileges)" on page 1053	Grants privileges on packages in the database.
"GRANT (role)" on page 1056	Grants roles to users, groups, or to other roles.
"GRANT (routine privileges)" on page 1059	Grants privileges on a routine (function, method, or procedure).
"GRANT (schema privileges)" on page 1063	Grants privileges on a schema.

Table 1. SQL schema statements (continued)

SQL Statement	Purpose
"GRANT (security label)" on page 1066	Grants a label-based access control (LBAC) security label for read access, write access, or for both read and write access.
"GRANT (sequence privileges)" on page 1069	Grants privileges on a sequence.
"GRANT (server privileges)" on page 1072	Grants privileges to query a specific data source.
"GRANT (SETSESSIONUSER privilege)" on page 1074	Grants the privilege to use the SET SESSION AUTHORIZATION statement.
"GRANT (table space privileges)" on page 1076	Grants privileges on a table space.
"GRANT (table, view, or nickname privileges)" on page 1078	Grants privileges on tables, views and nicknames.
"GRANT (workload privileges)" on page 1084	Grants the USAGE privilege on a workload.
"GRANT (XSR object privileges)" on page 1086	Grants the USAGE privilege on an XSR object.
"REFRESH TABLE" on page 1136	Refreshes the data in a materialized query table.
"RENAME" on page 1143	Renames an existing table.
"RENAME STOGROUP" on page 1145	Renames an existing storage group.
"RENAME TABLESPACE" on page 1146	Renames an existing table space.
"REVOKE (database authorities)" on page 1154	Revokes authorities from the entire database.
"REVOKE (exemption)" on page 1158	Revokes the exemption on an access rule for a specified label-based access control (LBAC) security policy.
"REVOKE (global variable privileges)" on page 1160	Revokes one or more privileges on a created global variable.
"REVOKE (index privileges)" on page 1162	Revokes the CONTROL privilege on given indexes.
"REVOKE (module privileges)" on page 1164	Revokes privileges on a module.
"REVOKE (package privileges)" on page 1166	Revokes privileges from given packages in the database.
"REVOKE (role)" on page 1169	Revokes roles from users, groups, or other roles.
"REVOKE (routine privileges)" on page 1171	Revokes privileges on a routine (function, method, or procedure).
"REVOKE (schema privileges)" on page 1175	Revokes privileges on a schema.
"REVOKE (security label)" on page 1177	Revokes a label-based access control (LBAC) security label for read access, write access, or for both read and write access.
"REVOKE (sequence privileges)" on page 1179	Revokes privileges on a sequence.
"REVOKE (server privileges)" on page 1181	Revokes privileges to query a specific data source.
"REVOKE (SETSESSIONUSER privilege)" on page 1183	Revokes the privilege to use the SET SESSION AUTHORIZATION statement.
"REVOKE (table space privileges)" on page 1185	Revokes the USE privilege on a given table space.
"REVOKE (table, view, or nickname privileges)" on page 1187	Revokes privileges from given tables, views or nicknames.
"REVOKE (workload privileges)" on page 1192	Revokes the USAGE privilege on a workload.
"REVOKE (XSR object privileges)" on page 1194	Revokes the USAGE privilege on an XSR object.

SQL statements

Table 1. SQL schema statements (continued)

SQL Statement	Purpose
"SET INTEGRITY" on page 1257	Sets the set integrity pending state and checks data for constraint violations.
"TRANSFER OWNERSHIP" on page 1306	Transfers ownership of a database object.

Table 2. SQL data change statements

SQL Statement	Purpose
"DELETE" on page 947	Deletes one or more rows from a table.
"INSERT" on page 1091	Inserts one or more rows into a table.
"MERGE" on page 1110	Updates a target (a table or view) using data from a source (result of a table reference).
"TRUNCATE" on page 1318	Deletes all rows from a table.
"UPDATE" on page 1321	Updates the values of one or more columns in one or more rows of a table.

Table 3. SQL data statements

SQL Statement	Purpose
"ALLOCATE CURSOR" on page 24	Allocates a cursor for the result set identified by the result set locator variable.
"ASSOCIATE LOCATORS" on page 268	Gets the result set locator value for each result set returned by a procedure.
"CLOSE" on page 287	Closes a cursor.
"DECLARE CURSOR" on page 928	Defines an SQL cursor.
"FETCH" on page 1019	Assigns values of a row to host variables.
"FLUSH BUFFERPOOLS" on page 1023	Writes out the dirty pages in the buffer pools to disk.
"FLUSH EVENT MONITOR" on page 1024	Writes out the active internal buffer of an event monitor.
"FLUSH FEDERATED CACHE" on page 1025	The FLUSH FEDERATED CACHE statement flushes the federated cache, allowing fresh metadata to be obtained the next time an SQL statement is issued against the remote table or view using a federated three part name.
"FLUSH OPTIMIZATION PROFILE CACHE" on page 1027	Removes the cached optimization profiles.
"FLUSH PACKAGE CACHE" on page 1029	Removes all cached dynamic SQL statements currently in the package cache.
"FREE LOCATOR" on page 1033	Removes the association between a locator variable and its value.
"LOCK TABLE" on page 1106	Either prevents concurrent processes from changing a table or prevents concurrent processes from using a table.
"OPEN" on page 1122	Prepares a cursor that will be used to retrieve values when the FETCH statement is issued.
"SELECT INTO" on page 1202	Specifies a result table of no more than one row and assigns the values to host variables.
"SET variable" on page 1291	Assigns values to variables.
"VALUES INTO" on page 1338	Specifies a result table of no more than one row and assigns the values to host variables.

Table 4. SQL transaction statements

SQL Statement	Purpose
"COMMIT" on page 300	Terminates a unit of work and commits the database changes made by that unit of work.
"RELEASE SAVEPOINT" on page 1142	Releases a savepoint within a transaction.
"ROLLBACK" on page 1195	Terminates a unit of work and backs out the database changes made by that unit of work.
"SAVEPOINT" on page 1198	Sets a savepoint within a transaction.

Table 5. SQL connection statements

SQL Statement	Purpose
"CONNECT (type 1)" on page 329	Connects to an application server according to the rules for remote unit of work.
"CONNECT (type 2)" on page 336	Connects to an application server according to the rules for application-directed distributed unit of work.
"DISCONNECT" on page 966	Terminates one or more connections when there is no active unit of work.
"RELEASE (connection)" on page 1140	Places one or more connections in the release-pending state.
"SET CONNECTION" on page 1207	Changes the state of a connection from dormant to current, making the specified location the current server.

Table 6. SQL dynamic statements

SQL Statement	Purpose
"DESCRIBE" on page 957	Obtains information about an object.
"DESCRIBE INPUT" on page 958	Obtains information about the input parameter markers of a prepared statement.
"DESCRIBE OUTPUT" on page 962	Obtains information about a prepared statement or information about the select list columns in a prepared SELECT statement.
"EXECUTE" on page 1003	Executes a prepared SQL statement.
"EXECUTE IMMEDIATE" on page 1011	Prepares and executes an SQL statement.
"PREPARE" on page 1130	Prepares an SQL statement (with optional parameters) for execution.

Table 7. SQL session statements

SQL Statement	Purpose
"DECLARE GLOBAL TEMPORARY TABLE" on page 934	Defines a declared temporary table.
"EXPLAIN" on page 1014	Captures information about the chosen access plan.
"SET COMPILATION ENVIRONMENT" on page 1206	Changes the current compilation environment in the connection to match the values contained in the compilation environment provided by a deadlock event monitor.
"SET CURRENT DECFLOAT ROUNDING MODE" on page 1209	Verifies that the specified rounding mode is the value that is currently set for the CURRENT DECFLOAT ROUNDING MODE special register.
"SET CURRENT DEFAULT TRANSFORM GROUP" on page 1211	Changes the value of the CURRENT DEFAULT TRANSFORM GROUP special register.
"SET CURRENT DEGREE" on page 1212	Changes the value of the CURRENT DEGREE special register.

Table 7. SQL session statements (continued)

SQL Statement	Purpose
"SET CURRENT EXPLAIN MODE" on page 1214	Changes the value of the CURRENT EXPLAIN MODE special register.
"SET CURRENT EXPLAIN SNAPSHOT" on page 1217	Changes the value of the CURRENT EXPLAIN SNAPSHOT special register.
"SET CURRENT FEDERATED ASYNCHRONY" on page 1219	Changes the value of the CURRENT FEDERATED ASYNCHRONY special register.
"SET CURRENT IMPLICIT XMLPARSE OPTION" on page 1221	Changes the value of the CURRENT IMPLICIT XMLPARSE OPTION special register.
"SET CURRENT ISOLATION" on page 1222	Changes the value of the CURRENT ISOLATION special register.
"SET CURRENT LOCALE LC_MESSAGES" on page 1223	Changes the value of the CURRENT LOCALE LC_MESSAGES special register.
"SET CURRENT LOCALE LC_TIME" on page 1225	Changes the value of the CURRENT LOCALE LC_TIME special register.
"SET CURRENT LOCK TIMEOUT" on page 1227	Changes the value of the CURRENT LOCK TIMEOUT special register.
"SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION" on page 1229	Changes the value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.
"SET CURRENT MDC ROLLOUT MODE" on page 1231	Assigns a value to the CURRENT MDC ROLLOUT MODE special register.
"SET CURRENT OPTIMIZATION PROFILE" on page 1233	Assigns a value to the CURRENT OPTIMIZATION PROFILE special register.
"SET CURRENT PACKAGE PATH" on page 1236	Assigns a value to the CURRENT PACKAGE PATH special register.
"SET CURRENT PACKAGESET" on page 1240	Sets the schema name for package selection.
"SET CURRENT QUERY OPTIMIZATION" on page 1242	Changes the value of the CURRENT QUERY OPTIMIZATION special register.
"SET CURRENT REFRESH AGE" on page 1245	Changes the value of the CURRENT REFRESH AGE special register.
"SET CURRENT SQL_CCFLAGS" on page 1247	Changes the value of the CURRENT SQL_CCFLAGS special register.
"SET CURRENT TEMPORAL BUSINESS_TIME" on page 1249	Changes the value of the CURRENT TEMPORAL BUSINESS_TIME special register.
"SET CURRENT TEMPORAL SYSTEM_TIME" on page 1251	Changes the value of the CURRENT TEMPORAL SYSTEM_TIME special register.
"SET ENCRYPTION PASSWORD" on page 1253	Sets the password for encryption.
"SET EVENT MONITOR STATE" on page 1255	Activates or deactivates an event monitor.
"SET PASSTHRU" on page 1276	Opens a session for submitting data source native SQL directly to the data source.
"SET PATH" on page 1278	Changes the value of the CURRENT PATH special register.
"SET ROLE" on page 1280	Verifies that the authorization ID of the session is a member of a specific role.
"SET SCHEMA" on page 1281	Changes the value of the CURRENT SCHEMA special register.
"SET SERVER OPTION" on page 1283	Sets server option settings.

Table 7. SQL session statements (continued)

SQL Statement	Purpose
"SET SESSION AUTHORIZATION" on page 1285	Changes the value of the SESSION USER special register.
"SET USAGE LIST STATE" on page 1288	Manages the state of a usage list and the associated data and memory.

Table 8. SQL embedded host language statements

SQL Statement	Purpose
"BEGIN DECLARE SECTION" on page 274	Marks the beginning of a host variable declaration section.
"END DECLARE SECTION" on page 1002	Marks the end of a host variable declaration section.
"GET DIAGNOSTICS" on page 1034	Used to obtain information about the previously executed SQL statement.
"INCLUDE" on page 1089	Inserts code or declarations into a source program.
"RESIGNAL" on page 1149	Used to resignal an error or warning condition.
"SIGNAL" on page 1303	Used to signal an error or warning condition.
"WHENEVER" on page 1341	Defines actions to be taken on the basis of SQL return codes.

Table 9. SQL control statements

SQL Statement	Purpose
"CALL" on page 276	Calls a procedure.
"CASE" on page 284	Selects an execution path based on multiple conditions.
"Compound SQL" on page 302	Encloses SQL statements with BEGIN and END keywords.
"Compound SQL (inlined)" on page 303	Combines one or more other SQL statements into an dynamic block.
"Compound SQL (embedded)" on page 308	Combines one or more other SQL statements into an executable block.
"Compound SQL (compiled)" on page 312	Groups other statements together in an SQL procedure.
"FOR" on page 1030	Executes a statement or group of statements for each row of a table.
"GOTO" on page 1037	Used to branch to a user-defined label within an SQL procedure.
"IF" on page 1087	Selects an execution path based on the evaluation of a condition.
"ITERATE" on page 1102	Causes the flow of control to return to the beginning of a labelled loop.
"LEAVE" on page 1104	Transfers program control out of a loop or a compound statement.
"LOOP" on page 1108	Repeats the execution of a statement or a group of statements.
"PIPE" on page 1128	Returns a row from a compiled table function.
"REPEAT" on page 1147	Executes a statement or group of statements until a search condition is true.
"RESIGNAL" on page 1149	Used to resignal an error or warning condition.
"RETURN" on page 1152	Used to return from a routine.
"SIGNAL" on page 1303	Used to signal an error or warning condition.
"WHILE" on page 1344	Repeats the execution of a statement or group of statements while a specified condition is true.

How SQL statements are invoked

SQL statements are classified as executable or non-executable.

An *executable statement* can be invoked in four ways. It can be:

- Issued interactively
- Prepared and executed dynamically
- Embedded in an application program
- Embedded in an SQL procedure, trigger, compound SQL (compiled), or compound SQL (inlined) with some restrictions:
  - Refer to "SQL-procedure-statement" in "Compound SQL (compiled)" on page 312 for the set of executable statements supported in SQL procedures and compound SQL (compiled) statements.
  - Refer to "SQL-statement" in "Compound SQL (inlined)" on page 303 statement for the set of executable statements supported in compound SQL (inlined) statements.
  - Refer to "SQL-procedure-statement" in "CREATE TRIGGER" on page 788 for the set of executable statements supported in a trigger.

Depending on the statement, some or all of these methods can be used. Statements embedded in REXX are prepared and executed dynamically.

A *non-executable statement* can only be embedded in an application program.

Another SQL statement construct is the select-statement. A *select-statement* can be invoked in three ways. It can be:

- Issued interactively
- Prepared dynamically, referenced in DECLARE CURSOR, and executed implicitly by OPEN, FETCH and CLOSE (dynamic invocation)
- Included in DECLARE CURSOR, and executed implicitly by OPEN, FETCH and CLOSE (static invocation)

Embedding a statement in an application program

SQL statements can be included in a source program that will be submitted to a precompiler. Such statements are said to be *embedded* in the program.

An embedded statement can be placed anywhere in the program where a host language statement is allowed. Each embedded statement must be preceded by the keywords EXEC SQL.

An executable statement embedded in an application program is executed every time a statement of the host language would be executed if it were specified in the same place. Thus, a statement within a loop is executed every time the loop is executed, and a statement within a conditional construct is executed only when the condition is satisfied.

An embedded statement can contain references to host variables. A host variable referenced in this way can be used in two ways. It can be used:

- As input (the current value of the host variable is used in the execution of the statement)
- As output (the variable is assigned a new value as a result of executing the statement)

In particular, all references to host variables in expressions and predicates are effectively replaced by current values of the variables; that is, the variables are used as input.

All executable statements should be followed by a test of the SQL return code. Alternatively, the WHENEVER statement (which is itself non-executable) can be used to change the flow of control immediately after the execution of an embedded statement.

All objects referenced in data manipulation language (DML) statements must exist when the statements are bound to a database.

An embedded non-executable statement is processed only by the precompiler. The precompiler reports any errors encountered in the statement. The statement is *never* processed during program execution; therefore, such statements should not be followed by a test of the SQL return code.

Statements can be included in the SQL-procedure-body portion of the CREATE PROCEDURE statement. Such statements are said to be embedded in the SQL procedure. Whenever an SQL statement description refers to a *host-variable*, an *SQL-variable* can be used if the statement is embedded in an SQL procedure.

Dynamic preparation and execution

An application program can dynamically build an SQL statement in the form of a character string placed in a host variable.

In general, the statement is built from some data available to the program (for example, input from a workstation). The statement (not a select-statement) constructed can be prepared for execution by means of the (embedded) PREPARE statement, and executed by means of the (embedded) EXECUTE statement. Alternatively, an (embedded) EXECUTE IMMEDIATE statement can be used to prepare and execute the statement in one step.

A statement that is going to be dynamically prepared must not contain references to host variables. It can instead contain parameter markers. (For rules concerning parameter markers, see “PREPARE”.) When the prepared statement is executed, the parameter markers are effectively replaced by current values of the host variables specified in the EXECUTE statement. Once prepared, a statement can be executed several times with different values for the host variables. Parameter markers are not allowed in the EXECUTE IMMEDIATE statement.

Successful or unsuccessful execution of the statement is indicated by the setting of an SQL return code in the SQLCA after the EXECUTE (or EXECUTE IMMEDIATE) statement completes. The SQL return code should be checked, as previously described. For more information, see “Detecting and processing error and warning conditions in host language applications” on page 12.

Static invocation of a select-statement

A select-statement can be included as a part of the (non-executable) DECLARE CURSOR statement.

Such a statement is executed every time the cursor is opened by means of the (embedded) OPEN statement. After the cursor is open, the result table can be retrieved, one row at a time, by successive executions of the FETCH statement.

Used in this way, the select-statement can contain references to host variables. These references are effectively replaced by the values that the variables have when the OPEN statement executes.

Dynamic invocation of a select-statement

An application program can dynamically build a select-statement in the form of a character string placed in a host variable.

In general, the statement is built from some data available to the program (for example, a query obtained from a workstation). The statement so constructed can be prepared for execution by means of the (embedded) PREPARE statement, and referenced by a (non-executable) DECLARE CURSOR statement. The statement is then executed every time the cursor is opened by means of the (embedded) OPEN statement. After the cursor is open, the result table can be retrieved, one row at a time, by successive executions of the FETCH statement.

Used in this way, the select-statement must not contain references to host variables. It can contain parameter markers instead. The parameter markers are effectively replaced by the values of the host variables specified in the OPEN statement.

Interactive invocation

A capability for entering SQL statements from a workstation is part of the architecture of the database manager. A statement entered in this way is said to be issued interactively.

Such a statement must be an executable statement that does not contain parameter markers or references to host variables, because these make sense only in the context of an application program.

SQL use with other host systems

SQL statement syntax exhibits minor variations among different types of host systems (DB2 for z/OS, DB2 for i, DB2 for Linux, UNIX, and Windows).

Regardless of whether the SQL statements in an application are static or dynamic, it is important - if the application is meant to access different database host systems - to ensure that the SQL statements and precompile/bind options are supported on the database systems that the application will access.

Further information about SQL statements used in other host systems can be found in the *SQL Reference* manuals for DB2 for z/OS and DB2 for i.

Detecting and processing error and warning conditions in host language applications

An application program containing executable SQL statements can use either SQLCODE or SQLSTATE values to handle return codes from SQL statements.

There are two ways in which an application can get access to these values.

- Include a structure named SQLCA. The SQLCA includes an integer variable named SQLCODE and a character string variable named SQLSTATE. In REXX, an SQLCA is provided automatically. In other languages, an SQLCA can be obtained by using the INCLUDE SQLCA statement.

- If LANGLEVEL SQL92E is specified as a precompile option, a variable named SQLCODE or SQLSTATE can be declared in the SQL declare section of the program. If neither of these variables is declared in the SQL declare section, it is assumed that a variable named SQLCODE is declared elsewhere in the program. With LANGLEVEL SQL92E, the program should not have an INCLUDE SQLCA statement.

An SQLCODE is set by the database manager after each SQL statement executes. All database managers conform to the ISO/ANSI SQL standard, as follows:

- If SQLCODE = 0 and SQLWARN0 is blank, execution was successful.
- If SQLCODE = 100, "no data" was found. For example, a FETCH statement returned no data, because the cursor was positioned after the last row of the result table.
- If SQLCODE > 0 and not = 100, execution was successful with a warning.
- If SQLCODE = 0 and SQLWARN0 = 'W', execution was successful, but one or more warning indicators were set.
- If SQLCODE < 0, execution was not successful.

The meaning of SQLCODE values other than 0 and 100 is product-specific.

An SQLSTATE is set by the database manager after each SQL statement executes. Application programs can check the execution of SQL statements by testing SQLSTATE instead of SQLCODE. SQLSTATE provides common codes for common error conditions. Application programs can test for specific errors or classes of errors. The coding scheme is the same for all IBM® database managers, and is based on the ISO/ANSI SQL92 standard.

SQL comments

Static SQL statements can include host language or SQL comments. Dynamic SQL statements can include SQL comments.

There are two types of SQL comments:

simple comments
Simple comments are introduced by two consecutive hyphens (--) and end with the end of line.

bracketed comments
Bracketed comments are introduced by /* and end with */.

The following rules apply to the use of simple comments:

- The two hyphens must be on the same line and must not be separated by a space.
- Simple comments can be started wherever a space is valid (except within a delimiter token or between 'EXEC' and 'SQL').
- Simple comments cannot be continued to the next line.
- In COBOL, the hyphens must be preceded by a space.

The following rules apply to the use of bracketed comments:

- The /* must be on the same line and must not be separated by a space.
- The */ must be on the same line and must not be separated by a space.
- Bracketed comments can be started wherever a space is valid (except within a delimiter token or between 'EXEC' and 'SQL').

- Bracketed comments can be continued to subsequent lines.

Examples

- *Example 1:* This example shows how to include simple comments in a statement:

```
CREATE VIEW PRJ_MAXPER       -- PROJECTS WITH MOST SUPPORT PERSONNEL
  AS SELECT PROJNO, PROJNAME -- NUMBER AND NAME OF PROJECT
    FROM PROJECT
    WHERE DEPTNO = 'E21'     -- SYSTEMS SUPPORT DEPT CODE
    AND PRSTAFF > 1
```

- *Example 2:* This example shows how to include bracketed comments in a statement:

```
CREATE VIEW PRJ_MAXPER       /* PROJECTS WITH MOST SUPPORT
                                   PERSONNEL                */
  AS SELECT PROJNO, PROJNAME /* NUMBER AND NAME OF PROJECT  */
    FROM PROJECT
    WHERE DEPTNO = 'E21'     /* SYSTEMS SUPPORT DEPT CODE   */
    AND PRSTAFF > 1
```

Conditional compilation in SQL

Conditional compilation allows SQL to include compiler directives which are used to determine the actual SQL that gets compiled.

There are two types of compiler directives that can be used for conditional compilation:

Selection directive
A compiler control statement used to determine the selection of a code fragment. The _IF directive can reference inquiry directives or global variables that are defined as a constant.

Inquiry directive
A reference to a compiler named constant that is assigned by the system or specified as a conditional compilation named constant in CURRENT SQL_CCFLAGS. An inquiry directive can be used directly or in a selection directive.

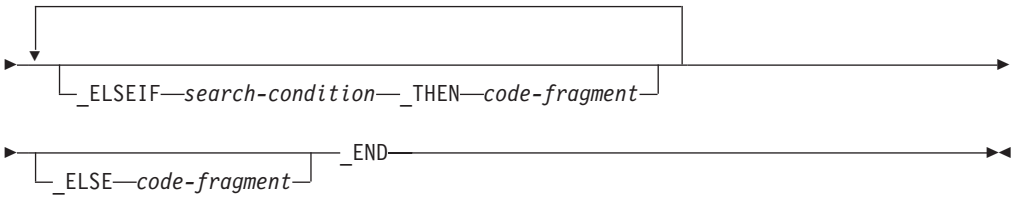
These directives can be used in the following contexts:

- SQL procedure definitions
- Compiled SQL function definitions
- Compiled trigger definitions
- Oracle PL/SQL package definitions

A directive can only appear after the object type (FUNCTION, PACKAGE, PACKAGE BODY, PROCEDURE, or TRIGGER) has been identified in the data definition language statement.

Selection directive

The selection directive is very similar to the IF statement except there are prefixes on the keywords to indicate use of conditional compilation and the terminating keyword is _END.



search-condition
Specifies the condition that is evaluated to determine what *code-fragment*, if any, is included. If the condition is unknown or false, evaluation continues with the next search condition, until a condition is true, the _ELSE clause is reached, or the end of the selection directive is reached. The search condition can include only the following elements (SQLSTATE 428HV):

- Constants of type BOOLEAN, INTEGER, or VARCHAR
- NULL constants
- Inquiry directives
- Global constants, where the defined constant value is a simple literal of type BOOLEAN, INTEGER, or VARCHAR
- Basic predicates
- NULL predicates
- Predicates that are a Boolean constant or a Boolean inquiry directive
- Logical operators (AND, OR, and NOT)

code-fragment
A portion of SQL code that can be included in the context of the SQL statement where the selection directive appears. There must not be a selection directive in *code-fragment* (SQLSTATE 428HV).

Inquiry directive

An inquiry directive is used to inquire about the compilation environment. An inquiry directive is specified in an SQL statement as an ordinary identifier prefixed with two underscore characters. The actual identifier can represent one of the following values:

- A compilation environment value defined by the system
- A compilation value defined by a user at the database level or at the individual session level

The only compilation environment variable defined by the system is __SQL_LINE, which provides the line number of SQL that is currently being compiled.

A user-defined compilation value can be defined at the database level using the **sql_ccflags** database configuration parameter or at a session level by assigning it to the CURRENT SQL_CCFLAGS special register.

If an inquiry directive is referenced but is not defined, processing continues assuming that the value for the inquiry directive is the null value.

Notes

- **References to global variables defined as constants:** A reference to a global variable (which can also be a reference to a module variable published in a

module) in a selection directive is used to provide a value based on a constant at the time of compilation only. The referenced global variable must meet the following requirements:

- Exist at the current server (SQLSTATE 42704)
- Have a data type of BOOLEAN, INTEGER, or VARCHAR (SQLSTATE 428HV)
- Be defined using the CONSTANT clause with a single constant value (SQLSTATE 428HV)

Such a global variable is known as a global constant. Subsequent changes to the global constant do not have any impact on statements that are already compiled.

- **Syntax alternatives:** If the data server environment is enabled for PL/SQL statement execution:
  - ELSIF can be specified instead of ELSEIF
  - A dollar character ($) can be used instead of an underscore character (_) as the prefix for the keywords for conditional compilation
  - Two dollar characters ($$) can be used instead of two underscore characters (__) as the prefix for an inquiry directive

  The dollar character prefix is intended only to support existing SQL statements that use inquiry directives and is not recommended for use when writing new SQL statements.

Example

Specify a database-wide setting for a compilation value called DBV97 that has a value of TRUE.

```
UPDATE DATABASE CONFIGURATION USING SQL_CCFLAGS DB2V97:TRUE
```

The value is available as the default for any subsequent connection to the database.

If a particular session needed a maximum number of years compilation value for use in defining some routines in the current session, the default SQL_CCFLAGS can be extended using the SET CURRENT SQL_CCFLAGS statement.

```
BEGIN
  DECLARE CCFLAGS_LIST VARCHAR(1024);
  SET CCFLAGS_LIST = CURRENT SQL_CCFLAGS CONCAT ',max_years:50';
  SET CURRENT SQL_CCFLAGS = CCFLAGS_LIST;
END
```

The use of CURRENT SQL_CCFLAGS on the right side of the assignment to the CCFLAGS_LIST variable keeps the existing SQL_CCFLAGS settings, while the string constant provides the additional compilation values.

Here is an example of a CREATE PROCEDURE statement that uses the contents of the CURRENT SQL_CCFLAGS.

```
CREATE PROCEDURE CHECK_YEARS (IN YEARS_WORKED INTEGER)
BEGIN
   _IF __DB2V97 _THEN
      IF YEARS_WORKED > __MAX_YEARS THEN
      ...
      END IF;
   _END
```

The inquiry directive __DB2V97 is used as a Boolean value to determine if the code can be included. The inquiry directive __MAX_YEARS is replaced during compilation by the constant value 50.

About SQL control statements

SQL control statements, also called SQL Procedural Language (SQL PL), are SQL statements that allow SQL to be used in a manner similar to writing a program in a structured programming language.

SQL control statements provide the capability to control the logic flow, declare, and set variables, and handle warnings and exceptions. Some SQL control statements include other nested SQL statements. SQL control statements can be used in the body of a routine, trigger or a compound statement.

References to SQL parameters, SQL variables, and global variables

SQL parameters, SQL variables, and global variables can be referenced anywhere in an SQL procedure statement where an expression or variable can be specified.

Host variables cannot be specified in SQL routines, SQL triggers or dynamic compound statements. SQL parameters can be referenced anywhere in the routine body, and can be qualified with the routine name. SQL variables can be referenced anywhere in the compound statement in which they are declared, and can be qualified with the label name specified at the beginning of the compound statement. If an SQL parameter or SQL variable has a row data type, fields can be referenced anywhere an SQL parameter or SQL variable can be referenced. Global variables can be referenced within any expression as long as the expression is not required to be deterministic. The following scenarios require deterministic expressions, which preclude the use of global variables:

- Check constraints
- Definitions of generated columns
- Refresh immediate MQTs

All SQL parameters, SQL variables, row variable fields, and global variables are considered nullable. The name of an SQL parameter, SQL variable, row variable field, or global variable in an SQL routine can be the same as the name of a column in a table or view referenced in the routine. The name of an SQL variable or row variable field can also be the same as the name of another SQL variable or row variable field declared in the same routine. This can occur when the two SQL variables are declared in different compound statements. The compound statement that contains the declaration of an SQL variable determines the scope of that variable. For more information, see “Compound SQL (Procedure)”.

The name of an SQL variable or SQL parameter in an SQL routine can be the same as the name of an identifier used in certain SQL statements. If the name is not qualified, the following rules describe whether the name refers to the identifier or to the SQL parameter or SQL variable:

- In the SET PATH and SET SCHEMA statements, the name is checked as an SQL parameter or SQL variable. If not found as an SQL variable or SQL parameter, it is used as an identifier.
- In the CONNECT, DISCONNECT, RELEASE, and SET CONNECTION statements, the name is used as an identifier.

Names that are the same should be explicitly qualified. Qualifying a name clearly indicates whether the name refers to a column, SQL variable, SQL parameter, row variable field, or global variable. If the name is not qualified, or qualified but still ambiguous, the following rules describe whether the name refers to a column, an SQL variable, an SQL parameter, or a global variable:

- If the tables and views specified in an SQL routine body exist at the time the routine is created, the name is first checked as a column name. If not found as a column, it is then checked as an SQL variable in the compound statement, then checked as an SQL parameter, and then, finally, checked as a global variable.
- If the referenced tables or views do not exist at the time the routine is created, the name is first checked as an SQL variable in the compound statement, then as an SQL parameter, and then as a global variable. The variable can be declared within the compound statement that contains the reference, or within a compound statement in which that compound statement is nested. If two SQL variables are within the same scope and have the same name, which can happen if they are declared in different compound statements, the SQL variable that is declared in the innermost compound statement is used. If not found, it is assumed to be a column.

References to SQL labels

Labels can be specified on most SQL procedure statements.

The compound statement that contains the statement that defines a label determines the scope of that label name. A label name must be unique within the compound statement in which it is defined, including any labels defined in compound statements that are nested within that compound statement (SQLSTATE 42734). The label must not be the same as a label specified on the compound statement itself (SQLSTATE 42734), or the same as the name of the routine that contains the SQL procedure statement (SQLSTATE 42734).

A label name can only be referenced within the compound statement in which it is defined, including any compound statements that are nested within that compound statement. A label can be used to qualify the name of an SQL variable, or it can be specified as the target of a GOTO, LEAVE, or ITERATE statement.

References to SQL condition names

The name of an SQL condition can be the same as the name of another SQL condition declared in the same routine.

This can occur when the two SQL conditions are declared in different compound statements. The compound statement that contains the declaration of an SQL condition name determines the scope of that condition name. A condition name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A condition name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a condition name, the condition that is declared in the innermost compound statement is the condition that is used. For more information, see “Compound SQL (inlined)”.

References to SQL statement names

The name of an SQL statement can be the same as the name of another SQL statement declared in the same routine.

This can occur when the two SQL statements are declared in different compound statements. The compound statement that contains the declaration of an SQL statement name determines the scope of that statement name. A statement name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A statement name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a statement name, the statement that is declared in the innermost compound statement is the statement that is used. For more information, see "Compound SQL (inlined)".

References to SQL cursor names

Cursor names include the names of declared cursors and the names of cursor variables.

The name of an SQL cursor can be the same as the name of another SQL cursor declared in the same routine. This can occur when the two SQL cursors are declared in different compound statements.

The compound statement that contains the declaration of an SQL cursor determines the scope of that cursor name. A cursor name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A cursor name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a cursor name, the cursor that is declared in the innermost compound statement is the cursor that is used. For more information, see "Compound SQL (inlined)".

If the cursor constructor assigned to a cursor variable contains a reference to a local SQL variable, then any OPEN statement that uses the cursor variable must be within the scope where the local SQL variable was declared.

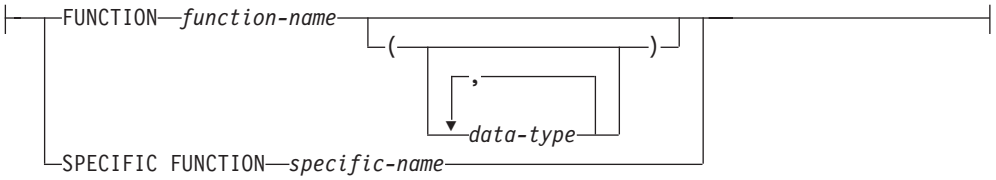
Function, method, and procedure designators

This topic describes syntax fragments that are used to uniquely identify a function, method, or procedure that is not defined in a module.

Function designator

A function designator uniquely identifies a single function. Function designators typically appear in DDL statements for functions (such as DROP or ALTER). A function designator must not identify a module function (SQLSTATE 42883).

function-designator:



FUNCTION *function-name*
Identifies a particular function, and is valid only if there is exactly one function instance with the name *function-name* in the schema. The identified function can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one instance of the function in the named or implied schema, an error (SQLSTATE 42725) is raised.

FUNCTION *function-name (data-type,...)*
Provides the function signature, which uniquely identifies the function. The function resolution algorithm is not used.

function-name
Specifies the name of the function. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Values must match the data types that were specified (in the corresponding position) on the CREATE FUNCTION statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific function instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because 0 < *n* < 25 means REAL, and 24 < *n* < 54 means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

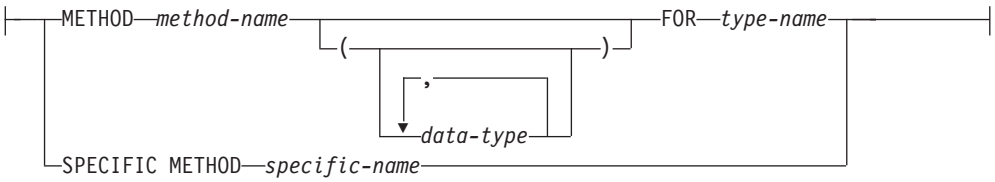
If no function with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *specific-name*
Identifies a particular user-defined function, using the name that is specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

Method designator

A method designator uniquely identifies a single method. Method designators typically appear in DDL statements for methods (such as DROP or ALTER).

method-designator:



METHOD *method-name*
Identifies a particular method, and is valid only if there is exactly one method instance with the name *method-name* for the type *type-name*. The identified method can have any number of parameters defined for it. If no method by this name exists for the type, an error (SQLSTATE 42704) is raised. If there is more than one instance of the method for the type, an error (SQLSTATE 42725) is raised.

METHOD *method-name (data-type,...)*
Provides the method signature, which uniquely identifies the method. The method resolution algorithm is not used.

method-name
Specifies the name of the method for the type *type-name*.

(data-type,...)
Values must match the data types that were specified (in the corresponding position) on the CREATE TYPE statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific method instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because $0 < n < 25$ means REAL, and $24 < n < 54$ means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the type in the named or implied schema, an error (SQLSTATE 42883) is raised.

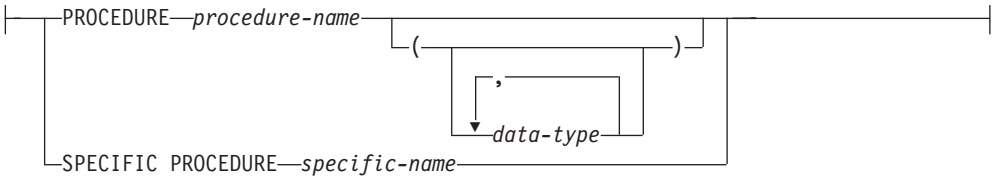
FOR *type-name*
Names the type with which the specified method is to be associated. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

SPECIFIC METHOD *specific-name*
Identifies a particular method, using the name that is specified or defaulted to at method creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific method instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

Procedure designator

A procedure designator uniquely identifies a single procedure. Procedure designators typically appear in DDL statements for procedures (such as DROP or ALTER). A procedure designator must not identify a module procedure (SQLSTATE 42883).

procedure-designator:



PROCEDURE *procedure-name*
Identifies a particular procedure, and is valid only if there is exactly one procedure instance with the name *procedure-name* in the schema. The identified procedure can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema,

an error (SQLSTATE 42704) is raised. If there is more than one instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is raised.

PROCEDURE *procedure-name (data-type,...)*
Provides the procedure signature, which uniquely identifies the procedure. The procedure resolution algorithm is not used.

procedure-name
Specifies the name of the procedure. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Values must match the data types that were specified (in the corresponding position) on the CREATE PROCEDURE statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific procedure instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because $0 < n < 25$ means REAL, and $24 < n < 54$ means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC PROCEDURE *specific-name*
Identifies a particular procedure, using the name that is specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

ALLOCATE CURSOR

The ALLOCATE CURSOR statement allocates a cursor for the result set identified by the result set locator variable.

For more information about result set locator variables, see the description of the ASSOCIATE LOCATORS statement.

Invocation

This statement can only be embedded in an SQL procedure. It is not an executable statement and cannot be dynamically prepared.

Authorization

None required.

Syntax

Description

cursor-name
Names the cursor. The name must not identify a cursor that has already been declared in the source SQL procedure (SQLSTATE 24502).

CURSOR FOR RESULT SET *rs-locator-variable*

Names a result set locator variable that has been declared in the source SQL procedure, according to the rules for declaring result set locator variables. For more information about declaring SQL variables, see "Compound SQL (Procedure) statement".

The result set locator variable must contain a valid result set locator value, as returned by the ASSOCIATE LOCATORS SQL statement (SQLSTATE 0F001).

Rules

- The following rules apply when using an allocated cursor:
  - An allocated cursor cannot be opened with the OPEN statement (SQLSTATE 24502).
  - An allocated cursor cannot be used in a positioned UPDATE or DELETE statement (SQLSTATE 42828).
  - An allocated cursor can be closed with the CLOSE statement. Closing an allocated cursor closes the associated cursor.
  - Only one cursor can be allocated to each result set.
- Allocated cursors last until a rollback operation, an implicit close, or an explicit close.
- A commit operation destroys allocated cursors that are not defined WITH HOLD.
- Destroying an allocated cursor closes the associated cursor in the SQL procedure.

Example

This SQL procedure example defines and associates cursor C1 with the result set locator variable LOC1 and the related result set returned by the SQL procedure:

```
ALLOCATE C1 CURSOR FOR RESULT SET LOC1;
```

ALTER AUDIT POLICY

The ALTER AUDIT POLICY statement modifies the definition of an audit policy at the current server.

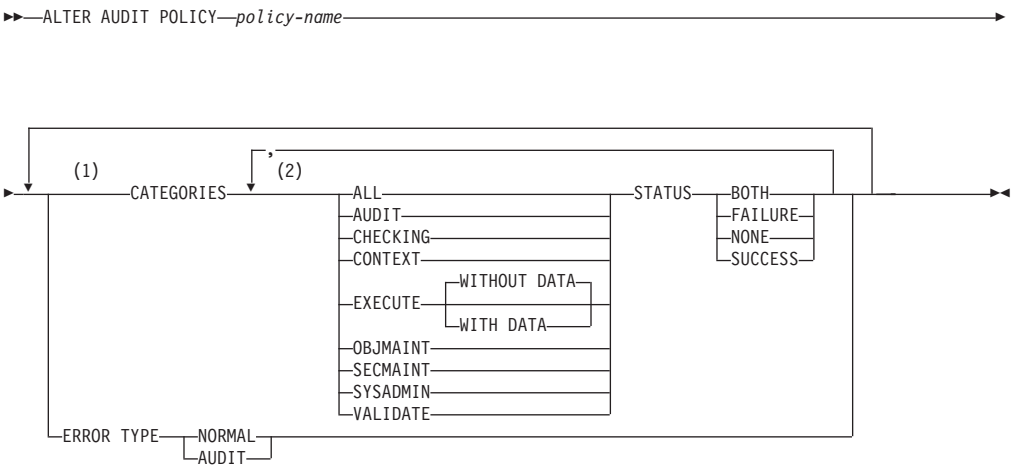
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Notes:

1 Each of the CATEGORIES and ERROR TYPE clauses can be specified at most once (SQLSTATE 42614).

2 Each category can be specified at most once (SQLSTATE 42614), and no other category can be specified if ALL is specified (SQLSTATE 42601).

Description

policy-name
Identifies the audit policy that is to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must uniquely identify an existing audit policy at the current server (SQLSTATE 42704).

CATEGORIES
A list of one or more audit categories for which a new status value is specified. If ALL is not specified, the STATUS of any category that is not explicitly specified remains unchanged.

ALL
Sets all categories to the same status. The EXECUTE category is WITHOUT DATA.

AUDIT
Generates records when audit settings are changed or when the audit log is accessed.

CHECKING
Generates records during authorization checking of attempts to access or manipulate database objects or functions.

CONTEXT
Generates records to show the operation context when a database operation is performed.

EXECUTE
Generates records to show the execution of SQL statements.

WITHOUT DATA or WITH DATA
Specifies whether or not input data values provided for any host variables and parameter markers should be logged as part of the EXECUTE category.

WITHOUT DATA
Input data values provided for any host variables and parameter markers are not logged as part of the EXECUTE category.

WITH DATA
Input data values provided for any host variables and parameter markers are logged as part of the EXECUTE category. Not all input values are logged; specifically, LOB, LONG, XML, and structured type parameters appear as the null value. Date, time, and timestamp fields are logged in ISO format. The input data values are converted to the database code page before being logged. If code page conversion fails, no errors are returned and the unconverted data is logged.

OBJMAINT
Generates records when data objects are created or dropped.

SECMAINT
Generates records when object privileges, database privileges, or DBADM authority is granted or revoked. Records are also generated when the database manager security configuration parameters **sysadm_group**, **sysctrl_group**, or **sysmaint_group** are modified.

SYSADMIN
Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.

VALIDATE
Generates records when users are authenticated or when system security information related to a user is retrieved.

STATUS
Specifies a status for the specified category.

BOTH
Successful and failing events will be audited.

FAILURE
Only failing events will be audited.

SUCCESS
Only successful events will be audited.

NONE
No events in this category will be audited.

ERROR TYPE
Specifies whether audit errors are to be returned or ignored.

NORMAL
Any errors generated by the audit are ignored and only the SQLCODEs for errors associated with the operation being performed are returned to the application.

AUDIT
All errors, including errors occurring within the audit facility itself, are returned to the application.

Rules

- An AUDIT-exclusive SQL statement must be followed by a COMMIT or ROLLBACK statement (SQLSTATE 5U021). AUDIT-exclusive SQL statements are:
  - AUDIT
  - CREATE AUDIT POLICY, ALTER AUDIT POLICY, or DROP (AUDIT POLICY)
  - DROP (ROLE) or DROP (TRUSTED CONTEXT) if the role or trusted context is associated with an audit policy
- An AUDIT-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Only one uncommitted AUDIT-exclusive SQL statement is allowed at a time across all database partitions. If an uncommitted AUDIT-exclusive SQL statement is executing, subsequent AUDIT-exclusive SQL statements wait until the current AUDIT-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.
- If the audit policy that is being altered is currently associated with a database object, the changes do not take effect until the next unit of work for the application that is affected by the change. For example, if the audit policy is in use for the database, no current units of work will see the change to the policy until after a COMMIT or a ROLLBACK statement for that unit of work completes.

Example

Alter the SECMAINT, CHECKING, and VALIDATE categories of an audit policy named DBAUDPRF to audit both successes and failures.

```
ALTER AUDIT POLICY DBAUDPRF
  CATEGORIES SECMAINT STATUS BOTH,
             CHECKING STATUS BOTH,
             VALIDATE STATUS BOTH
```

ALTER BUFFERPOOL

The ALTER BUFFERPOOL statement is used to modify the characteristics or behavior of a buffer pool.

The ALTER BUFFERPOOL statement can modify a buffer pool in the following ways:

- Modify the size of the buffer pool on all members or on a single member
- Enable or disable automatic sizing of the buffer pool
- Add this buffer pool definition to a new database partition group
- Modify the block area of the buffer pool for block-based I/O

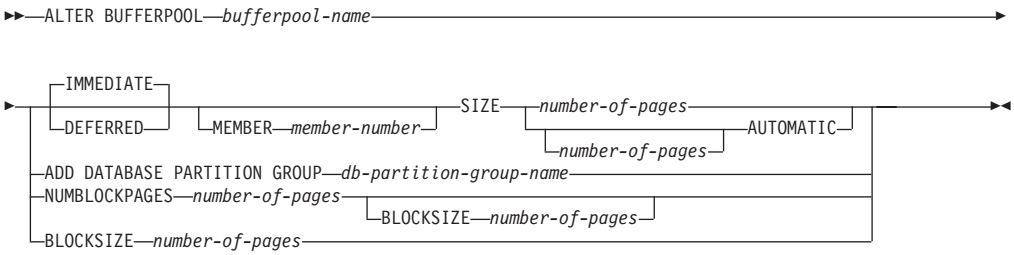
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax



Description

bufferpool-name
Names the buffer pool. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a buffer pool described in the catalog.

IMMEDIATE or DEFERRED
Indicates whether or not the buffer pool size will be changed immediately.

IMMEDIATE
The buffer pool size will be changed immediately. If there is not enough reserved space in the database shared memory to allocate new space (SQLSTATE 01657), the statement is executed as DEFERRED.

DEFERRED
The buffer pool size will be changed when the database is reactivated (all applications need to be disconnected from the database). Reserved memory space is not needed; the DB2 database will allocate the required memory from the system at activation time.

MEMBER *member-number*
Specifies the member on which the size of the buffer pool is modified. An exception entry is created in the SYSCAT.BUFFERPOOLEXCEPTIONS catalog

view. The member must be in one of the database partition groups for the buffer pool (SQLSTATE 42729). If this clause is not specified, the size of the buffer pool is modified on all members except those that have an exception entry in SYSCAT.BUFFERPOOLEXCEPTIONS.

SIZE
Specifies a new size for the buffer pool, or enables or disables self tuning for this buffer pool.

number-of-pages
The number of pages for the new buffer pool size. If the buffer pool is already a self-tuning buffer pool, and the SIZE *number-of-pages* clause is specified, the alter operation disables self-tuning for this buffer pool.

AUTOMATIC
Enables self tuning for this buffer pool. The database manager adjusts the size of the buffer pool in response to workload requirements. If the number of pages is specified, the current buffer pool size is set to that value unless the deferred keyword is also specified, in which case the number of pages will be ignored. Note that the self-tuning memory manager (STMM) enforces a minimum size for automatic buffer pools, and that any specified size is a one-time setting - on subsequent database activations, the buffer pool size is based on the last tuning value. To determine the current size of buffer pools that are enabled for self tuning, use the **GET SNAPSHOT** command and examine the current size of the buffer pools (the value of the **bp_cur_buffsz** monitor element). When AUTOMATIC is specified, the MEMBER clause cannot be specified (SQLSTATE 42601).

ADD DATABASE PARTITION GROUP *db-partition-group-name*
Adds this database partition group to the list of database partition groups to which the buffer pool definition is applicable. For any member in the database partition group that does not already have the buffer pool defined, the buffer pool is created on the member using the default size specified for the buffer pool. Table spaces in *db-partition-group-name* may specify this buffer pool. The database partition group must currently exist in the database (SQLSTATE 42704).

NUMBLOCKPAGES *number-of-pages*
Specifies the number of pages that should exist in the block-based area. The number of pages must not be greater than 98 percent of the number of pages for the buffer pool (SQLSTATE 54052). Specifying the value 0 disables block I/O. The actual value of NUMBLOCKPAGES used will be a multiple of BLOCKSIZE.

NUMBLOCKPAGES is not supported in a DB2 pureScale® environment (SQLSTATE 56038).

BLOCKSIZE *number-of-pages*
Specifies the number of pages in a block. The block size must be a value between 2 and 256 (SQLSTATE 54053). The default value is 32.

BLOCKSIZE is not supported in a DB2 pureScale environment (SQLSTATE 56038).

Notes

- Only the buffer pool size can be changed dynamically (immediately). All other changes are deferred, and will only come into effect after the database is reactivated.

- If the statement is executed as deferred, although the buffer pool definition is transactional and the changes to the buffer pool definition will be reflected in the catalog tables on commit, no changes to the actual buffer pool will take effect until the next time the database is started. The current attributes of the buffer pool will exist until then, and there will not be any impact to the buffer pool in the interim. Tables created in table spaces of new database partition groups will use the default buffer pool. The statement is IMMEDIATE by default when that keyword applies.
- There should be enough real memory on the machine for the total of all the buffer pools, as well as for the rest of the database manager and application requirements.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DBPARTITIONNUM or NODE can be specified in place of MEMBER except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

ALTER DATABASE PARTITION GROUP

The ALTER DATABASE PARTITION GROUP statement is used to add one or more database partitions to a database partition group, or drop one or more database partitions from a database partition group.

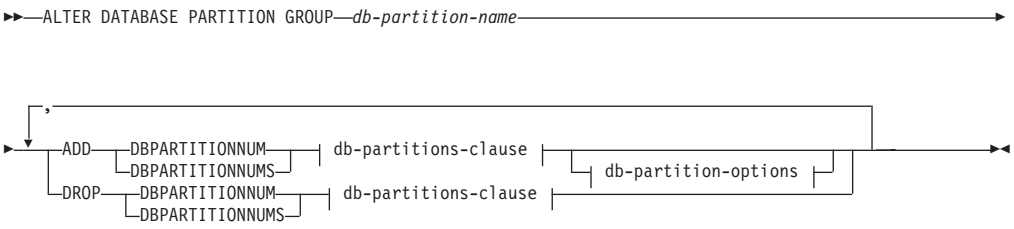
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

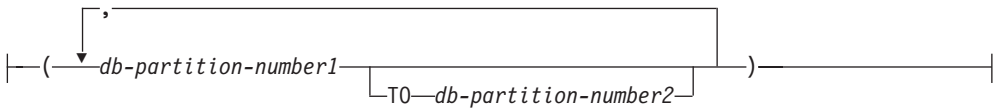
Authorization

The authorization ID of the statement must have SYSCTRL or SYSADM authority.

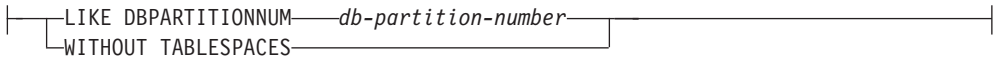
Syntax



db-partitions-clause:



db-partition-options:



Description

db-partition-name
Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a database partition group described in the catalog. IBMCATGROUP and IBMTEMPGROUP cannot be specified (SQLSTATE 42832).

ADD DBPARTITIONNUM
Specifies the specific database partition or partitions to add to the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must not already be defined in the database partition group (SQLSTATE 42728).

DROP DBPARTITIONNUM
Specifies the specific database partition or partitions to drop from the database

partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must already be defined in the database partition group (SQLSTATE 42729).

db-partitions-clause
Specifies the database partition or partitions to be added or dropped.

db-partition-number1
Specify a specific database partition number.

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9).

db-partition-options

LIKE DBPARTITIONNUM *db-partition-number*
Specifies that the containers for the existing table spaces in the database partition group will be the same as the containers on the specified *db-partition-number*. The specified database partition must be a partition that existed in the database partition group before this statement, and that is not included in a DROP DBPARTITIONNUM clause of the same statement.

For table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all), the containers will not necessarily match those from the specified partition. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database, and this might or might not result in the same containers being used. The size of each table space is based on the initial size that was specified when the table space was created, and might not match the current size of the table space on the specified partition.

WITHOUT TABLESPACES
Specifies that the containers for existing table spaces in the database partition group are not created on the newly added database partition or partitions. The ALTER TABLESPACE statement using the *db-partitions-clause* or the MANAGED BY AUTOMATIC STORAGE clause must be used to define containers for use with the table spaces that are defined on this database partition group. If this option is not specified, the default containers are specified on newly added database partitions for each table space defined on the database partition group.

This option is ignored for table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all). There is no way to defer container creation for these table spaces. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. The size of each table space will be based on the initial size that was specified when the table space was created.

Rules

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).

- Each *db-partition-number* listed in the *db-partitions-clause* must be for a unique database partition (SQLSTATE 42728).
- A valid database partition number is between 0 and 999 inclusive (SQLSTATE 42729).
- A database partition cannot appear in both the ADD and DROP clauses (SQLSTATE 42728).
- There must be at least one database partition remaining in the database partition group. The last database partition cannot be dropped from a database partition group (SQLSTATE 428C0).
- If neither the LIKE DBPARTITIONNUM clause nor the WITHOUT TABLESPACES clause is specified when adding a database partition, the default is to use the lowest database partition number of the existing database partitions in the database partition group (say it is *2*) and proceed as if LIKE DBPARTITIONNUM *2* had been specified. For an existing database partition to be used as the default, it must have containers defined for all the table spaces in the database partition group (column IN_USE of SYSCAT.DBPARTITIONGROUPDEF is not 'T').
- The ALTER DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Notes

- When a database partition is added to a database partition group, a catalog entry is made for the database partition (see SYSCAT.DBPARTITIONGROUPDEF). The distribution map is changed immediately to include the new database partition, along with an indicator (IN_USE) that the database partition is in the distribution map if either:
  - no table spaces are defined in the database partition group or
  - no tables are defined in the table spaces defined in the database partition group and the WITHOUT TABLESPACES clause was not specified.

  The distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is not included in the distribution map if either:
  - Tables exist in table spaces in the database partition group or
  - Table spaces exist in the database partition group and the WITHOUT TABLESPACES clause was specified (unless all of the table spaces are defined to use automatic storage, in which case the WITHOUT TABLESPACES clause is ignored)

  To change the distribution map, the REDISTRIBUTE DATABASE PARTITION GROUP command must be used. This redistributes any data, changes the distribution map, and changes the indicator. Table space containers need to be added before attempting to redistribute data if the WITHOUT TABLESPACES clause was specified.
- When a database partition is dropped from a database partition group, the catalog entry for the database partition (see SYSCAT.DBPARTITIONGROUPDEF) is updated. If there are no tables defined in the table spaces defined in the database partition group, the distribution map is changed immediately to exclude the dropped database partition and the entry for the database partition in the database partition group is dropped. If tables exist, the distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is waiting to be dropped. The REDISTRIBUTE DATABASE PARTITION

GROUP command must be used to redistribute the data and drop the entry for the database partition from the database partition group.

- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NODE can be specified in place of DBPARTITIONNUM
  - NODES can be specified in place of DBPARTITIONNUMS
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Example

Assume that you have a six-partition database that has the following database partitions: 0, 1, 2, 5, 7, and 8. Two database partitions (3 and 6) are added to the system.

- *Example 1*: Assume that you want to add database partitions 3 and 6 to a database partition group called MAXGROUP, and have table space containers like those on database partition 2. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MAXGROUP
  ADD DBPARTITIONNUMS (3,6)LIKE DBPARTITIONNUM 2
```

- *Example 2*: Assume that you want to drop database partition 1 and add database partition 6 to database partition group MEDGROUP. You will define the table space containers separately for database partition 6 using ALTER TABLESPACE. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MEDGROUP
  ADD DBPARTITIONNUM(6)WITHOUT TABLESPACES
  DROP DBPARTITIONNUM(1)
```

ALTER DATABASE

The ALTER DATABASE statement adds new storage paths to, or removes existing storage paths from, the collection of paths that are used for automatic storage table spaces.

An automatic storage table space is a table space that has been created using automatic storage; that is, the MANAGED BY AUTOMATIC STORAGE clause has been specified on the CREATE TABLESPACE statement, or no MANAGED BY clause has been specified at all. If a database is enabled for automatic storage, container and space management characteristics of its table spaces can be completely determined by the database manager. If the database is not currently enabled for automatic storage then the act of adding storage paths will enable it.

Important: This statement is deprecated and might be removed in a future release. Use the CREATE STOGROUP or ALTER STOGROUP statements instead.

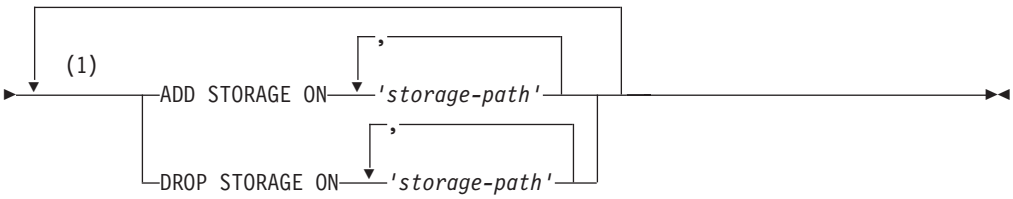
Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include either SYSADM or SYSCTRL authority.

Syntax



Notes:

1 Each clause can be specified only once.

Description

database-name
An optional value specifying the name of the database that is to be altered. If specified, the value must match the name of the database to which the application is currently connected (not the alias that the client might have cataloged); otherwise, an error is returned (SQLSTATE 42961).

ADD STORAGE ON
Specifies that one or more new storage paths are to be added to the collection of storage paths that are used for automatic storage table spaces.

'storage-path'
A string constant that specifies either an absolute path or the letter name of a drive (Windows operating systems only) on which containers for automatic storage table spaces are to be created.

DROP STORAGE ON
Specifies that one or more storage paths are to be removed from the collection of storage paths that are used for automatic storage table spaces. If table spaces are actively using a storage path being dropped, then the state of the storage path is changed from "In Use" to "Drop Pending" and future use of the storage path will be prevented.

'storage-path'
A string constant that specifies either an absolute path or the letter name of a drive (Windows operating systems only).

Rules

- For a database that is running on Version 10.1 or later, the operations of this statement are applied to the default storage group for the database. If no storage group is defined for the database, the name IBMSTOGROUP is used.
- A storage path being added, must be valid according to the naming rules for paths, and must be accessible (SQLSTATE 57019). Similarly, in a partitioned database environment, the storage path must exist and be accessible on every database partition (SQLSTATE 57019).
- A storage path being dropped must currently exist in the database (SQLSTATE 57019) and cannot already be in the "Drop Pending" state (SQLSTATE 55073).
- A database enabled for automatic storage must have at least one storage path. Dropping all storage paths from the database is not permitted (SQLSTATE 428HH).
- The ALTER DATABASE statement cannot be executed while a database partition server is being added (SQLSTATE 55071).
- DROP STORAGE ON cannot be specified in a DB2 pureScale environment (SQLSTATE 56038).

Notes

- When adding new storage paths:
  - Existing regular and large table spaces using automatic storage will not initially use these new paths. The database manager might choose to create new table space containers on these paths only if an out-of-space condition occurs.
  - Existing temporary table spaces managed by automatic storage do not automatically use new storage paths. The database must be stopped normally then restarted for containers in these table spaces to use the new storage path or paths. As an alternative, the temporary table spaces can be dropped and recreated. When created, these table spaces automatically use all storage paths that have sufficient free space.
- Adding storage paths to the database to enable automatic storage will not cause the database to convert existing non-automatic storage enabled table spaces to use automatic storage.
- Although ADD STORAGE and DROP STORAGE are logged operations, whether they are redone during a rollforward operation depends on how the database

was restored. If the restore operation does not redefine the storage paths that are associated with the database, the log record that contains the storage path change is redone, and the storage paths that are described in the log record are added or dropped during the rollforward operation. However, if the storage paths *are* redefined during the restore operation, the rollforward operation will not redo ADD STORAGE or DROP STORAGE log records, because it is assumed that you have already set up the storage paths.

- When free space is calculated for a storage path on a database partition, the database manager checks for the existence of the following directories or mount points within the storage path, and will use the first one that is found.

  ```
  <storage path>/<instance name>/NODE####/<database name>
  <storage path>/<instance name>/NODE####
  <storage path>/<instance name>
  <storage path>
  ```

  Where:

  – <storage path> is a storage path associated with the database
  – <instance name> is the instance under which the database resides
  – NODE#### corresponds to the database partition number (for example, NODE0000 or NODE0001)
  – <database name> is the name of the database

  File systems can be mounted at a point beneath the storage path, and the database manager will recognize that the actual amount of free space available for table space containers might not be the same amount that is associated with the storage path directory itself.

  Consider an example in which two logical database partitions exist on one physical machine, and there is a single storage path (`/db2data`). Each database partition will use this storage path, but you might want to isolate the data from each partition within its own file system. In this case, a separate file system can be created for each partition and it can be mounted at `/db2data/<instance>/NODE####`. When creating containers on the storage path and determining free space, the database manager will not retrieve free space information for `/db2data`, but instead will retrieve it for the corresponding `/db2data/<instance>/NODE####` directory.

- In general, the same storage paths must be used for each partition in a partitioned database environment. One exception to this is the case in which database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path, such that the resulting path name is different on each partition.
- When dropping a storage path that is in use by one or more table spaces, the state of the path changes from “In Use” to “Drop Pending”. Future growth on the path will not occur. Before the path can be fully removed from the database, each affected table space must be rebalanced (using the REBALANCE clause of the ALTER TABLESPACE statement) so that its container data is moved off the storage path. Rebalance is only supported for regular and large table spaces. Temporary table spaces should be dropped and recreated to have their containers removed from the dropped path. When the path is no longer in use by any table space, it will be physically removed from the database.

  For a partitioned database, the path is maintained independently on each partition. When a path is no longer in use on a given database partition, it will be physically removed from that partition. Other partitions may still show the path as being in the “Drop Pending” state.

  The list of automatic storage table spaces using drop pending storage paths can be determined by issuing the following SQL statement:

```
SELECT DISTINCT A.TBSP_NAME, A.TBSP_ID, A.TBSP_CONTENT_TYPE
  FROM SYSIBMADM.SNAPTBSP A, SYSIBMADM.SNAPTBSP_PART B
  WHERE A.TBSP_ID = B.TBSP_ID AND B.TBSP_PATHS_DROPPED = 1
```

- When dropping a storage path that was originally specified using a database partition expression, the same storage path string, including the database partition expression, must be used in the drop. If a database partition expression was specified then this path string can be found in the "Path with db partition expression" element (db_storage_path_with_dpe) of a database snapshot. This element is not shown if a database partition expression was not included in the original path specified.
- It is possible for a given storage path to be added to a database multiple times. When using the DROP STORAGE ON clause, specifying that particular path once will drop *all* instances of the path from the database.

Examples

- *Example 1:* Add two paths under the /db2 directory (/db2/filesystem1 and /db2/filesystem2) and a third path named /filesystem3 to the space for automatic storage table spaces that is associated with the currently connected database.

  ```
  ALTER DATABASE ADD STORAGE ON '/db2/filesystem1', '/db2/filesystem2',
    '/filesystem3'
  ```

- *Example 2:* Add drives D and E to the space for automatic storage table spaces that is associated with the SAMPLE database.

  ```
  ALTER DATABASE SAMPLE ADD STORAGE ON 'D:', 'E:\'
  ```

- *Example 3:* Add directory F:\DB2DATA and drive G to the space for automatic storage table spaces that is associated with the currently connected database.

  ```
  ALTER DATABASE ADD STORAGE ON 'F:\DB2DATA', 'G:'
  ```

- *Example 4:* Add a storage path that uses a database partition expression to differentiate the storage paths on each of the database partitions.

  ```
  ALTER DATABASE ADD STORAGE ON '/dataForPartition $N'
  ```

  The storage path that would be used on database partition 0 is /dataForPartition0; on database partition 1, it would be /dataForPartition1; and so on.

- *Example 5:* Add storage paths to a database that is not automatic storage enabled, for the purposes of enabling automatic storage for the database.

  ```
  CREATE DATABASE MYDB AUTOMATIC STORAGE NO
  CONNECT TO MYDB
  ALTER DATABASE ADD STORAGE ON '/db2/filesystem1', '/db2/filesystem2'
  ```

  Database MYDB is now enabled for automatic storage.

- *Example 6:* Remove paths /db2/filesystem1 and /db2/filesystem2 from the currently connected database.

  ```
  ALTER DATABASE DROP STORAGE ON '/db2/filesystem1', '/db2/filesystem2'
  ```

  After the storage is dropped successfully, use the ALTER TABLESPACE statement with the REBALANCE clause for each table space that was using these storage paths to rebalance the table space.

- *Example 7:* A storage path with a database partition expression (/dataForPartition $N) was previously added to the database and now it is to be removed.

  ```
  ALTER DATABASE DROP STORAGE ON '/dataForPartition $N'
  ```

After the storage is dropped successfully, use the ALTER TABLESPACE statement with the REBALANCE clause for each table space that was using these storage paths to rebalance the table space.

ALTER EVENT MONITOR

The ALTER EVENT MONITOR statement alters the definition of an event monitor that has a target for the event monitor data of TABLE.

Invocation

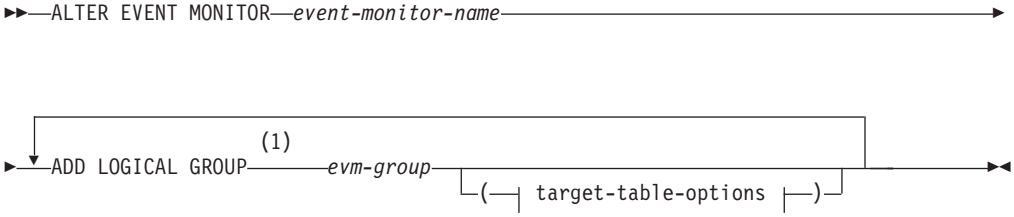
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

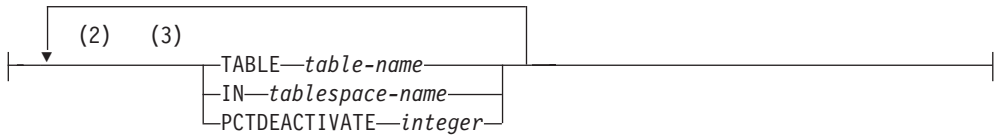
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority

Syntax



target-table-options:



Notes:

1 A logical group can be added only to TABLE event monitors (not UNFORMATTED EVENT TABLE event monitors).

2 Each clause can be specified only once.

3 Clauses can be separated with a space or a comma.

Description

event-monitor-name
The *event-monitor-name* must identify an event monitor that exists at the current server and has a target for the event monitor data of TABLE.

ADD LOGICAL GROUP
Adds a logical group to the event monitor that has a target for the data of TABLE.

evm-group
Identifies the logical data group for which a target table is being added. The value depends upon the type of event monitor, as shown in the following table:

Table 10. Values for evm-group based on the type of event monitor

Type of Event Monitor	evm-group Value
Database	• DB • CONTROL[1] • DBMEMUSE
Tables	• TABLE • CONTROL[1]
Deadlocks	• CONNHEADER • DEADLOCK • DLCONN • CONTROL[1]
Deadlocks with details	• CONNHEADER • DEADLOCK • DLCONN[2] • DLLOCK[3] • CONTROL[1]
Deadlocks with details history	• CONNHEADER • DEADLOCK • DLCONN[2] • DLLOCK[3] • STMTHIST • CONTROL[1]
Deadlocks with details history values	• CONNHEADER • DEADLOCK • DLCONN[2] • DLLOCK[3] • STMTHIST • STMTVALS • CONTROL[1]
Tablespaces	• TABLESPACE • CONTROL[1]
Bufferpools	• BUFFERPOOL • CONTROL[1]
Connections	• CONNHEADER • CONN • CONTROL[1] • CONNMEMUSE

Table 10. Values for evm-group based on the type of event monitor (continued)

Type of Event Monitor	evm-group Value
Statements	• CONNHEADER • STMT • SUBSECTION[4] • CONTROL[1]
Transactions	• CONNHEADER • XACT • CONTROL[1]
Activities	• ACTIVITY • ACTIVITYMETRICS • ACTIVITYSTMT • ACTIVITYVALS • CONTROL[1]
Statistics	• QSTATS • SCSTATS • SCMETRICS • WCSTATS • WLSTATS • WLMETRICS • HISTOGRAMBIN • CONTROL[1]
Threshold Violations	• THRESHOLDVIOLATIONS • CONTROL[1]
Locking[5]	• LOCK • LOCK_PARTICIPANTS • LOCK_PARTICIPANT_ACTIVITIES • LOCK_ACTIVITY_VALUES • CONTROL[1]
Package Cache[5]	• PKGCACHE • PKGCACHE_METRICS • CONTROL[1]
Unit of Work[5]	• UOW • UOW_METRICS • UOW_PACKGE_LIST • UOW_EXECUTABLE_LIST • CONTROL[1]

Table 10. Values for evm-group based on the type of event monitor (continued)

Type of Event Monitor	evm-group Value
Change History	• CHANGESUMMARY • EVMONSTART • TXNCOMPLETION • DDLSTMTEXEC • DBDBMCFG • REGVAR • UTILSTART • UTILSTOP • UTILPHASE • UTILLOCATION • CONTROL[1]

[1] Logical data groups dbheader (conn_time element only), start and overflow, are all written to the CONTROL group. The overflow group is written if the event monitor is non-blocked and events were discarded.

[2] Corresponds to the DETAILED_DLCONN event.

[3] Corresponds to the LOCK logical data groups that occur within each DETAILED_DLCONN event.

[4] Created only for partitioned database environments.

[5] Refers to the Formatted Event Table version of this event monitor type.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If no name is provided, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
substring(evm-group CONCAT '_'
   CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*
Defines the table space in which the table is to be created. If no table space name is provided, the table space is chosen using the same process as when a table is created without a table space name using the CREATE TABLE statement.

When specifying the table space name for an activities, locking, package cache, or unit of work event monitor, the table space's page size affects the INLINE LOB lengths used. Therefore, consider specifying a table space with as large a page size as possible to improve the INSERT performance of the event monitor.

PCTDEACTIVATE *integer*
If a table is being created in a DMS table space, PCTDEACTIVATE specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100. The default value is 100 (meaning that the event monitor deactivates when the table space becomes completely

full). This option is ignored for SMS table spaces. When a target table space has auto-resize enabled, it is recommended that PCTDEACTIVATE be set to 100.

Notes

- **When system catalog changes take effect:** Changes are written to the system catalog, but do not take effect until they are committed and the event monitor is reactivated.

Example

The event monitor ACT is missing the ACTIVITYMETRICS group. Alter the event monitor to add this group and give the table the name "ACTMETRICS".

```
ALTER EVENT MONITOR ACT
   ADD LOGICAL GROUP ACTIVITYMETRICS TABLE ACTMETRICS
```

ALTER FUNCTION

The ALTER FUNCTION statement modifies the properties of an existing function.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTERIN privilege on the schema of the function
- Owner of the function, as recorded in the OWNER column of the SYSCAT.ROUTINES catalog view
- DBADM authority

To alter the EXTERNAL NAME of a function, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

- CREATE_EXTERNAL_ROUTINE authority on the database
- DBADM authority

To alter a function to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

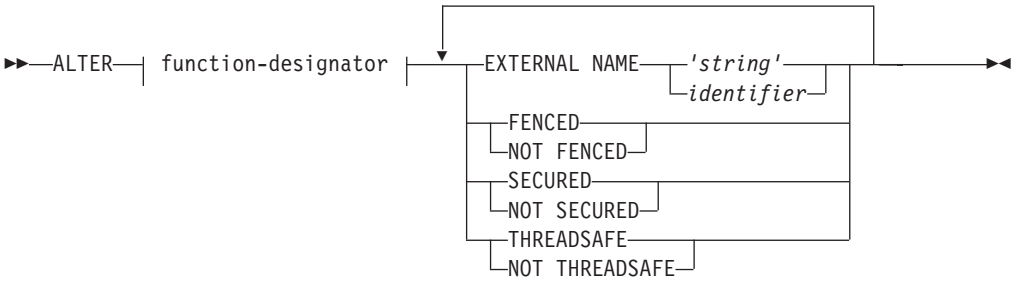
To alter a function to be fenced, no additional authorities or privileges are required.

To alter a function to be SECURED or NOT SECURED the privileges held by the authorization ID of the statement must include at least one of the following authorities:

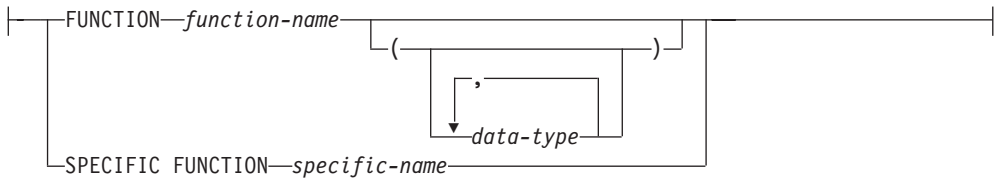
- SECADM authority
- CREATE_SECURE_OBJECT authority

If no other clauses are specified, then no other privileges are required to process the statement.

Syntax



function-designator:



Description

function-designator
Uniquely identifies the function to be altered. For more information, see "Function, method, and procedure designators" on page 20.

EXTERNAL NAME *'string'* **or** *identifier*
Identifies the name of the user-written code that implements the function. This option can only be specified when altering external functions (SQLSTATE 42849).

FENCED or NOT FENCED
Specifies whether the function is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most functions have the option of running as FENCED or NOT FENCED.

If a function is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the function. In general, a function running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for functions that were not adequately coded, reviewed, and tested can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED user-defined functions are used.

A function declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a function has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the function cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE, OLEDB, or CLR functions (SQLSTATE 42849).

SECURED or NOT SECURED
Specifies whether the function is considered secure for row and column access control.

NOT SECURED
Indicates that the function is not considered secure. When the function is invoked, the arguments of the function must not reference a column for which a column mask is enabled and column level access control is activated for its table (SQLSTATE 428HA). This rule applies to the non secure user-defined functions that are invoked anywhere in the statement.

SECURED
Indicates that the function is considered secure.

The function must be secure when it is referenced in a row permission or a column mask (SQLSTATE 428H8).

The function must be secure when it is referenced in a materialized query table and the materialized query table references any table that has row or column level access control activated (SQLSTATE 428H8).

THREADSAFE or NOT THREADSAFE
Specifies whether the function is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the function is defined with LANGUAGE other than OLE and OLEDB:

- If the function is defined as THREADSAFE, the database manager can invoke the function in the same process as other routines. In general, to be threadsafe, a function should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED functions can be THREADSAFE.
- If the function is defined as NOT THREADSAFE, the database manager will never simultaneously invoke the function in the same process as another routine. Only a fenced function can be NOT THREADSAFE (SQLSTATE 42613).

This option may not be altered for LANGUAGE OLE or OLEDB functions (SQLSTATE 42849).

Notes

- It is not possible to alter a function that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Functions declared as LANGUAGE SQL, sourced functions, or template functions cannot be altered (SQLSTATE 42917).
- ***Altering a function from NOT SECURED to SECURED***: Normally users with SECADM authority do not have privileges to alter database objects such as user-defined functions and triggers. Typically they will examine the actions taken by a function, ensure it is secure, then grant the CREATE_SECURE_OBJECT authority to someone who has required privileges to alter the user-defined function to be secure. After the function is altered, they will revoke the CREATE_SECURE_OBJECT authority from the user who was granted this authority.

  The function is considered secure. The SECURED attribute is considered to be an assertion that declares the user has established a change control audit procedure for all changes to the user-defined function. The database manager assumes that

such a control audit procedure is in place for all subsequent ALTER FUNCTION statements or changes to external packages.

Packages and dynamically cached SQL statements that depend on the function might be invalidated because the secure attribute affects the access path selection for statements involving tables for which row or column level access control is activated and the function being replaced.

- ***Altering a function from SECURED to NOT SECURED***: The function is considered not secure. Packages and dynamically cached SQL statements that depend on the function might be invalidated because the secure attribute affects the access path selection for statements involving tables for which row or column level access control is activated.
- ***Invoking other user-defined functions in a secure function***: When a secure user-defined function is referenced in a data manipulation statement where a row or column access control enforced table is referenced, if the secure user-defined function invokes other user-defined functions, the database manager does not validate whether those nested user-defined functions are secure. If those nested functions can access sensitive data, the user with SECADM authority needs to ensure those functions are allowed to access those data and a change control audit procedure has been established for all changes to those functions.

Example

The function MAIL() has been thoroughly tested. To improve its performance, alter the function to be not fenced.

```
ALTER FUNCTION MAIL() NOT FENCED
```

ALTER HISTOGRAM TEMPLATE

The ALTER HISTOGRAM TEMPLATE statement is used to modify the template describing the type of histogram that can be used to override one or more of the default histograms of a service class or a work class.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

Syntax

Description

template-name
Names the histogram template. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must identify an existing histogram template at the current server (SQLSTATE 42704). The template name can be the default system histogram template SYSDEFAULTHISTOGRAM.

HIGH BIN VALUE *bigint-constant*
Specifies the top value of the second to last bin (the last bin has an unbounded top value). The units depend on how the histogram is used. The maximum value is 268 435 456.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (WORK ACTION SET)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (WORK CLASS SET)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.

Example

Change the high bin value of a histogram template named LIFETIMETEMP.

```
ALTER HISTOGRAM TEMPLATE LIFETIMETEMP
  HIGH BIN VALUE 90000
```

ALTER INDEX

The ALTER INDEX statement alters the definition of an index.

Invocation

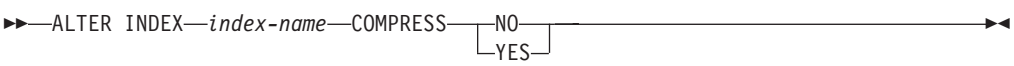
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTERIN privilege on the schema of the index
- ALTER privilege on the table on which the index is defined
- CONTROL privilege on the index
- DBADM authority

Syntax



Description

INDEX *index-name*
Identifies the index to be altered. The name must identify an index that exists at the current server (SQLSTATE 42704).

COMPRESS
Specifies whether index compression is to be enabled or disabled. The index must not be an MDC or ITC block index, catalog index, XML path index, index specification, or an index on a created temporary table or declared temporary table (SQLSTATE 42995).

NO Specifies that index compression is disabled. A compressed index will remain compressed until the index is rebuilt via index reorganization or recreation.

YES
Specifies that index compression is enabled. An uncompressed index will remain uncompressed until the index is rebuilt via index reorganization or recreation.

Example

Alter index JOB_BY_DPT to be compressed index.

```
ALTER INDEX JOB_BY_DPT
COMPRESS YES
```

ALTER MASK

The ALTER MASK statement alters a column mask that exists at the current server.

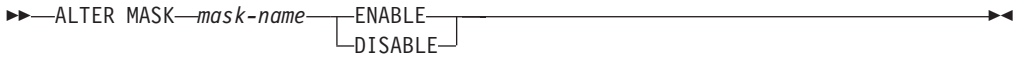
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

mask-name
Identifies the column mask to be altered. The name must identify a mask that exists at the current server (SQLSTATE 42704).

ENABLE

Enables the column mask. If column level access control is not currently activated on the table, the column mask will become effective when column level access control is activated on the table. If column level access control is currently activated on the table, the column mask becomes effective immediately and all packages and dynamically cached statements that reference the table are invalidated.

ENABLE is ignored if the column mask is already enabled.

DISABLE

Disables the column mask. If column level access control is not currently activated on the table, the column mask will remain ineffective when column level access control is activated on the table. If column level access control is currently activated on the table, the column mask becomes ineffective immediately and all packages and dynamically cached statements that reference the table are invalidated.

DISABLE is ignored if the column mask is already disabled.

Examples

- *Example 1*: Enable column mask M1.

```
ALTER MASK M1 ENABLE
```

- *Example 2*: Disable column mask M1.

```
ALTER MASK M1 DISABLE
```

ALTER METHOD

The ALTER METHOD statement modifies an existing method by changing the method body associated with the method.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

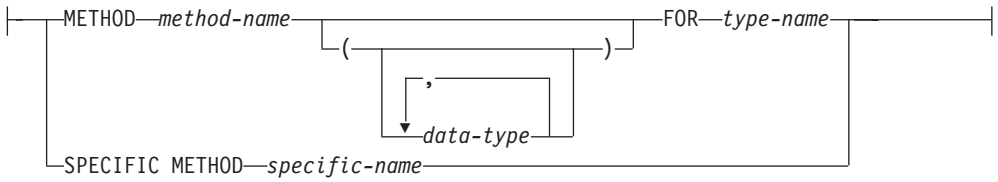
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CREATE_EXTERNAL_ROUTINE authority on the database, and at least one of:
  - ALTERIN privilege on the schema of the type
  - Owner of the type, as recorded in the OWNER column of the SYSCAT.DATATYPES catalog view
- DBADM authority

Syntax

method-designator:



Description

method-designator
Uniquely identifies the method to be altered. For more information, see "Function, method, and procedure designators" on page 20.

EXTERNAL NAME *'string'* **or** *identifier*
Identifies the name of the user-written code that implements the method. This option can only be specified when altering external methods (SQLSTATE 42849).

Notes

- It is not possible to alter a method that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Methods declared as LANGUAGE SQL cannot be altered (SQLSTATE 42917).
- Methods declared as LANGUAGE CLR cannot be altered (SQLSTATE 42849).

- The specified method must have a body before it can be altered (SQLSTATE 42704).

Example

Alter the method DISTANCE() in the structured type ADDRESS_T to use the library `newaddresslib`.

```
ALTER METHOD DISTANCE()
  FOR ADDRESS_T
  EXTERNAL NAME 'newaddresslib!distance2'
```

ALTER MODULE

The ALTER MODULE statement alters the definition of a module.

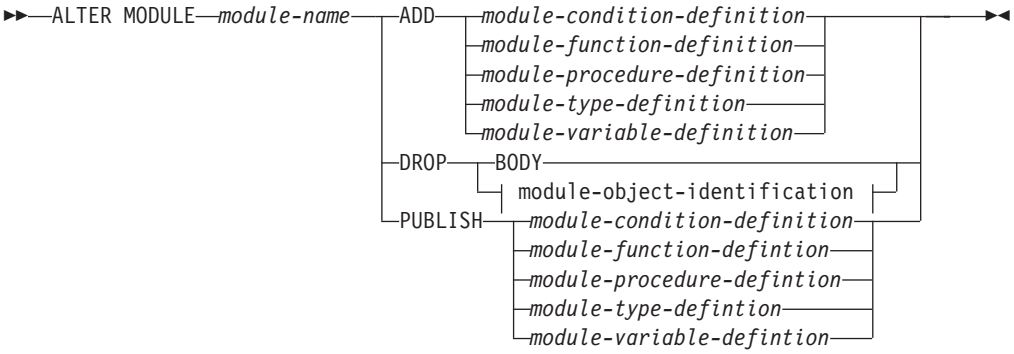
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

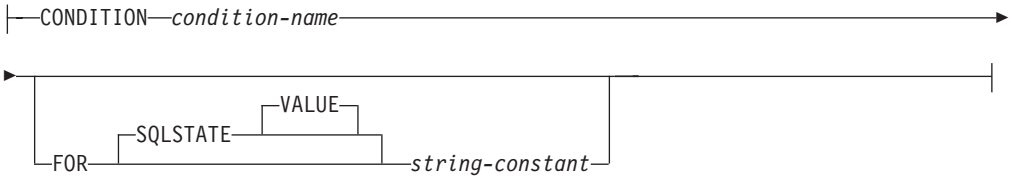
Authorization

The privileges held by the authorization ID of the statement must include ownership of the module and also include all of the privileges necessary to invoke the SQL statements that are specified within the ALTER MODULE statement.

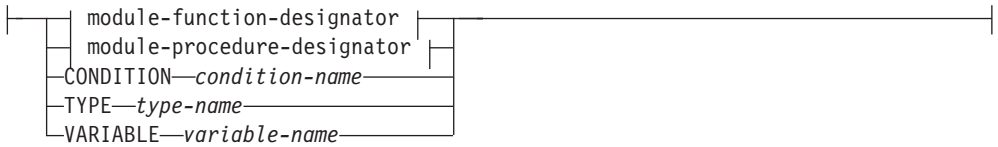
Syntax



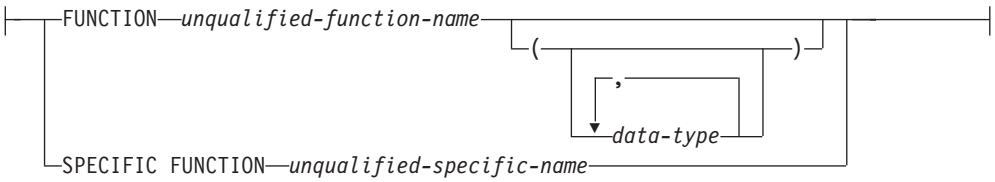
module-condition-definition:



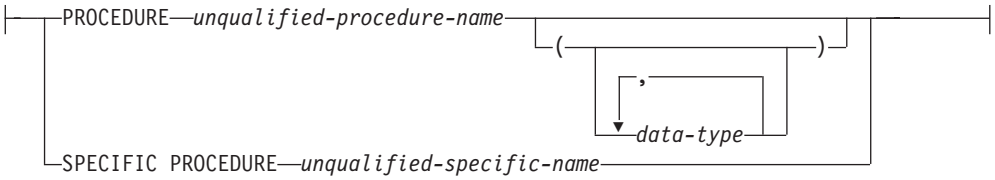
module-object-identification:



module-function-designator:



module-procedure-designator:



Description

module-name
Identifies the module to be altered. The module-name must identify a module that exists at the current server (SQLSTATE 42704).

ADD
Adds an object to the module or adds the body to a routine definition that already exists in the module without a body. If adding a user-defined type or a global variable, the object must not identify a user-defined type or global variable that already exists in the module. If the user-defined type or global variable did not exist, it is added to the module for use within the module only.

If adding a routine and the specified routine does not exist, the routine is added. If adding a routine and the specified routine exists, the existing routine definition must not include a routine body (SQLSTATE 42723). This routine prototype is completely replaced by the new routine definition, including the routine attributes and the routine body, except that the published attribute is retained. The specified routine is considered to exist if one of the following conditions is true:

- There is a routine in the module with the same specific name and same routine name.
- The specified routine is a procedure and there is a procedure in the module with the same procedure name and the same number of parameters. The names and data types of the parameters do not need to match.
- The specified routine is a function and there is a function in the module with the same function name and the same number of parameters with matching data types. The length, precision, and scale of parameter data types are not compared and can be different when determining if the specified routine exists. The names of the parameters do not need to match.

module-condition-definition
Adds a module condition.

condition-name
Name of the condition. The name must not identify an existing condition in the module. The condition-name must be specified without any qualification (SQLSTATE 42601). The name of the condition must be unique within the module.

FOR SQLSTATE *string-constant*
Specifies the SQLSTATE that is associated with the condition. The string-constant must be specified as five characters enclosed in single quotation marks, and the SQLSTATE class (the first two characters) must not be '00'. This is an optional clause.

module-function-definition
The syntax to add a function is the same as the CREATE FUNCTION statement excluding the CREATE keyword and both the function-name and specific-name must be specified without any qualification (SQLSTATE 42601). If the function is unique within the module, a new function is added. If the function matches an existing function that does not include a body (SQL-routine-body or EXTERNAL NAME clause), then this function prototype is replaced by the new definition except that the published attribute is retained. All SQL functions added to a module are processed as if a compound SQL (compiled) statement was used.

The module function definition can only specify the RETURNS TABLE clause when the SQL-routine-body is an compound SQL (compiled) statement that specifies NOT ATOMIC. The module function definition must not specify the SOURCE clause, the TEMPLATE clause, or the LANGUAGE OLEDEB option (SQLSTATE 42613).

module-procedure-definition
The syntax to define the procedure is the same as the CREATE PROCEDURE statement excluding the CREATE keyword and both the procedure-name and specific-name must be specified without any qualification (SQLSTATE 42601). If the procedure signature is unique within the module, a new procedure is added. If the procedure matches an existing procedure that does not include a body (SQL-routine-body or EXTERNAL NAME clause), then this procedure prototype is replaced by the new definition except that the published attribute is retained. The name of the procedure can begin with "SYS_" only to add the module initialization procedure called SYS_INIT. See Notes for details.

module-type-definition
The syntax to define the user-defined type is the same as the CREATE TYPE statement excluding the CREATE keyword and the type-name must be specified without any qualification (SQLSTATE 42601). The name of the user-defined type must be unique within the module. A structured type cannot be defined in a module. Any generated functions required to support the type definition are also defined in the module. If the module user-defined type is published then so are the generated functions.

module-variable-definition
The syntax to define the variable is the same as the CREATE VARIABLE statement excluding the CREATE keyword and the variable-name must be specified without any qualification (SQLSTATE 42601). The name of the variable must be unique within the module.

DROP
Drops a specified part of a module. The module-object-identification syntax is used to identify the object to be dropped unless the body of the module is being dropped.

BODY
Drops the module body, which includes:

- all objects that are not published.
- the routine body of any published SQL routines

- the EXTERNAL reference for any published external routines.

PUBLISH
Adds a new object to the module and makes it available for use outside the module. In the case of routines, a routine prototype can be specified that does not include the executable body of the routine.

module-condition-definition
Adds a module condition that is available for use outside the module.

condition-name
Name of the condition. The name must not identify an existing condition in the module. The condition-name must be specified without any qualification (SQLSTATE 42601). The name of the condition must be unique within the module.

FOR SQLSTATE *string-constant*
Specifies the SQLSTATE that is associated with the condition. The string-constant must be specified as five characters enclosed in single quotation marks, and the SQLSTATE class (the first two characters) must not be '00'. This is an optional clause.

module-function-definition
The syntax to define the function is the same as the CREATE FUNCTION statement excluding the CREATE keyword and both the function-name and specific-name must be specified without any qualification (SQLSTATE 42601). The definition of the function must include the function name, full specification of any parameters and the returns clause. Module user-defined data types that are not published are not candidates for the parameter data types or the RETURNS clause data type. Module variables that are not published are not candidates for the anchor object in an ANCHOR clause of a parameter data type or a returns data type. A function prototype can be specified by omitting the LANGUAGE clause (or specifying LANGUAGE SQL) and the SQL-routine-body. The function signature must be unique within the module. The name of the function must not begin with "SYS_" (SQLSTATE 42939). All SQL functions added to a module are processed as if a compound SQL (compiled) statement was used.

The module function definition can only specify the RETURNS TABLE clause when the SQL-routine-body is an compound SQL (compiled) statement that specifies NOT ATOMIC. The module function definition must not specify the SOURCE clause, the TEMPLATE clause, or the LANGUAGE OLEDEB option (SQLSTATE 42613).

module-procedure-definition
The syntax to define the procedure is the same as the CREATE PROCEDURE statement excluding the CREATE keyword and both the procedure-name and specific-name must be specified without any qualification (SQLSTATE 42601). The definition of the procedure must include the procedure name and full specification of any parameters. Module user-defined data types that are not published are not candidates for the parameter data types. Module variables that are not published are not candidates for the anchor object in an ANCHOR clause of a parameter definition. A function prototype can be specified by omitting the LANGUAGE clause (or specifying LANGUAGE SQL) and the SQL-routine-body. The procedure signature must be unique within the module. The name of the procedure must not begin with "SYS_" (SQLSTATE 42939).

module-type-definition
The syntax to define the user-defined type is the same as the CREATE TYPE statement excluding the CREATE keyword and the type-name must be specified without any qualification (SQLSTATE 42601). Module user-defined data types that are not published are not candidates for any data type referenced in the module user-defined data type definition. Module variables that are not published are not candidates for the anchor object in an ANCHOR clause. The name of the user-defined type must not begin with "SYS_" (SQLSTATE 42939) and must be unique within the module. A structured type cannot be defined in a module. Any generated functions required to support the type definition are also defined in the module as published functions.

module-variable-definition
The syntax to define the variable is the same as the CREATE VARIABLE statement excluding the CREATE keyword and the variable-name must be specified without any qualification (SQLSTATE 42601). Module user-defined data types that are not published are not candidates for the any data type referenced in the variable definition. Module variables that are not published are not candidates for the anchor object in an ANCHOR clause. The name of the variable must not begin with "SYS_" (SQLSTATE 42939) and must be unique within the module.

module-object-identification
Identifies a unique module object.

module-function-designator
Uniquely identifies a single module function.

FUNCTION *unqualified-function-name*
Identifies a particular function, and is valid only if there is exactly one function instance with the name unqualified-function-name in the module. The identified function can have any number of parameters defined for it. If no function by this name exists in the module, an error (SQLSTATE 42704) is raised. If there is more than one instance of the function in the module, an error (SQLSTATE 42725) is raised.

FUNCTION *unqualified-function-name (data type,...)*
Provides the function signature, which uniquely identifies the function. The function resolution algorithm is not used.

unqualified-function-name
Specifies the name of the function.

(data-type,...)
Values must match the data types that were specified (in the corresponding position) when the function was originally defined. The number of data types, and the logical concatenation of the data types, is used to identify the specific function instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match. FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or

DOUBLE). If length, precision, or scale is coded, the value must exactly match that specified when the function was defined.

A type of FLOAT(n) does not need to match the defined value for n, because 0 < n < 25 means REAL, and 24 < n < 54 means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE. If no function with the specified signature exists in the module, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *unqualified-specific-name*
Identifies a particular user-defined function, using the name that is specified or defaulted to at function definition time. The unqualified-specific-name must identify a specific function instance in the module; otherwise, an error is returned (SQLSTATE 42704).

module-procedure-designator
Uniquely identifies a single module procedure.

PROCEDURE *unqualified-procedure-name*
Identifies a particular procedure, and is valid only if there is exactly one procedure instance with the name unqualified-procedure-name in the module. The identified procedure can have any number of parameters defined for it. If no procedure by this name exists in the module, an error is returned (SQLSTATE 42704). If there is more than one instance of the procedure in the module, an error is returned (SQLSTATE 42725).

PROCEDURE *unqualified-procedure-name (data-type,...)*
Provides the procedure signature, which uniquely identifies the procedure. The procedure resolution algorithm is not used.

unqualified-procedure-name
Specifies the name of the procedure.

(data-type,...)
Values must match the data types that were specified (in the corresponding position) when the procedure was originally defined. The number of data types, and the logical concatenation of the data types, is used to identify the specific procedure instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE). If length, precision, or scale is coded, the value must exactly match that specified in when the procedure was defined.

A type of FLOAT(n) does not need to match the defined value for n, because 0 < n < 25 means REAL, and 24 < n < 54 means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the module, an error is returned (SQLSTATE 42883).

SPECIFIC PROCEDURE *unqualified-specific-name*
Identifies a particular procedure, using the name that is specified or defaulted to at procedure definition time. The unqualified-specific-name must identify a specific procedure instance in the module; otherwise, an error is returned (SQLSTATE 42704).

TYPE *type-name*
Identifies a user-defined type from the module. The type-name must be specified without any qualification (SQLSTATE 42601) and must identify a user-defined type that exists in the module (SQLSTATE 42704).

VARIABLE *variable-name*
Identifies a global variable from the module. The variable-name must be specified without any qualification (SQLSTATE 42601) and must identify a global variable that exists in the module (SQLSTATE 42704).

CONDITION *condition-name*
Identifies a condition from the module. The condition-name must be specified without any qualification and must identify a condition that exists in the module (SQLSTATE 42737).

Rules

- Names of objects in the module cannot begin with "SYS_" with the exception of specifically designated SYS_INIT procedure name (SQLSTATE 42939).
- **ALTER MODULE DROP FUNCTION**: If the function is referenced in the definition of a row permission or column mask, the function cannot be dropped (SQLSTATE 42893).
- **ALTER MODULE DROP VARIABLE**: If the variable is referenced in the definition of a row permission or column mask, the variable cannot be dropped (SQLSTATE 42893).
- **ALTER MODULE DROP BODY**: If the module is referenced in the definition of a row permission or column mask, the module cannot be dropped (SQLSTATE 42893).

Notes

- ***Module initialization*:** A module can have a special initialization procedure called SYS_INIT that is implicitly executed when the first reference is made to a module routine or module global variable. The SYS_INIT procedure must be implemented with no parameters, cannot return result sets, and can be an SQL or external procedure that cannot be published (SQLSTATE 428HP). If the SYS_INIT procedure fails, an error is returned for the statement that caused the module initialization (SQLSTATE 56098).
- ***Use of module conditions*:** A module condition can only be used with a SIGNAL statement, RESIGNAL statement or a handler declaration that is within a compound SQL (compiled) statement.
- ***Invalidation*:** If a routine prototype is replaced using the ADD action, all objects that depended on the published module routine are invalidated. If DROP BODY is issued, all objects dependent on published module routines are invalidated.
- ***Obfuscation*:** The ALTER MODULE ADD FUNCTION, ALTER MODULE ADD PROCEDURE, ALTER MODULE PUBLISH FUNCTION, and ALTER MODULE PUBLISH PROCEDURE statements can be submitted in obfuscated form. In an obfuscated statement, only the routine name and its parameters are readable.

The rest of the statement is encoded in such a way that is not readable but can be decoded by the database server. Obfuscated statements can be produced by calling the DBMS_DDL.WRAP function.

Example

The following statements create a module named INVENTORY containing an associative array type, a variable of that data type, a procedure that adds elements to the array and a function that extracts elements from the array. Only the function and the procedure can be referenced from outside of the module based on the PUBLISH keyword in the corresponding ALTER MODULE statements. The data type and the variable can only be referenced by other objects in the module.

```
CREATE MODULE INVENTORY

ALTER MODULE INVENTORY ADD
TYPE ITEMLIST AS INTEGER ARRAY[VARCHAR(100)]

ALTER MODULE INVENTORY ADD
VARIABLE ITEMS ITEMLIST

ALTER MODULE INVENTORY PUBLISH
PROCEDURE UPDATE_ITEM(NAME VARCHAR(100), QUANTITY INTEGER)
BEGIN
SET ITEMS[NAME] = QUANTITY;
END

ALTER MODULE INVENTORY PUBLISH
FUNCTION CHECK_ITEM(NAME VARCHAR(100)) RETURNS INTEGER
RETURN ITEMS[NAME]
```

ALTER NICKNAME

The ALTER NICKNAME statement modifies the nickname information associated with a data source object (such as a table, view, or file).

This statement modifies the information that is stored in the federated database in the following ways:

- Altering the local column names for the columns of the data source object
- Altering the local data types for the columns of the data source object
- Adding, setting, or dropping nickname and column options
- Adding or dropping a primary key
- Adding or dropping one or more unique, referential, or check constraints
- Altering one or more referential or check constraint attributes
- Altering the caching of data at a federated server

Invocation

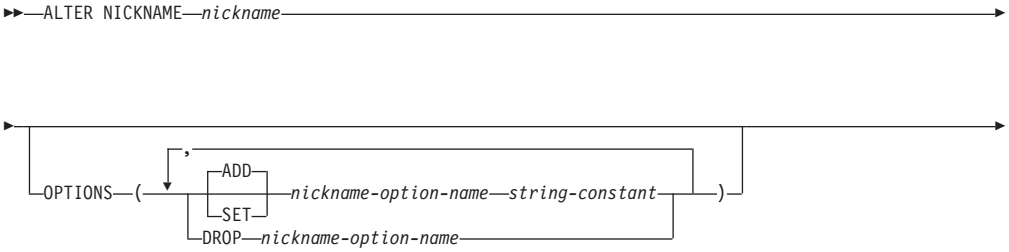
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

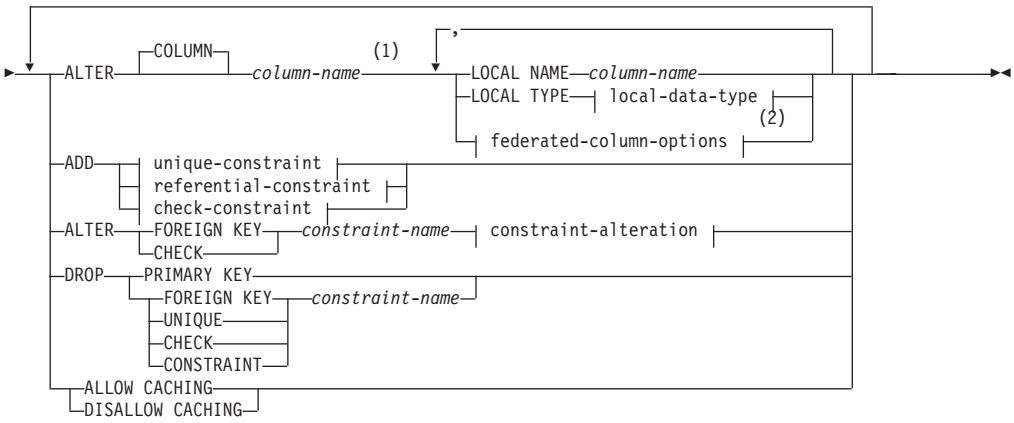
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Owner of the nickname, as recorded in the OWNER column of the SYSCAT.TABLES catalog view
- DBADM authority

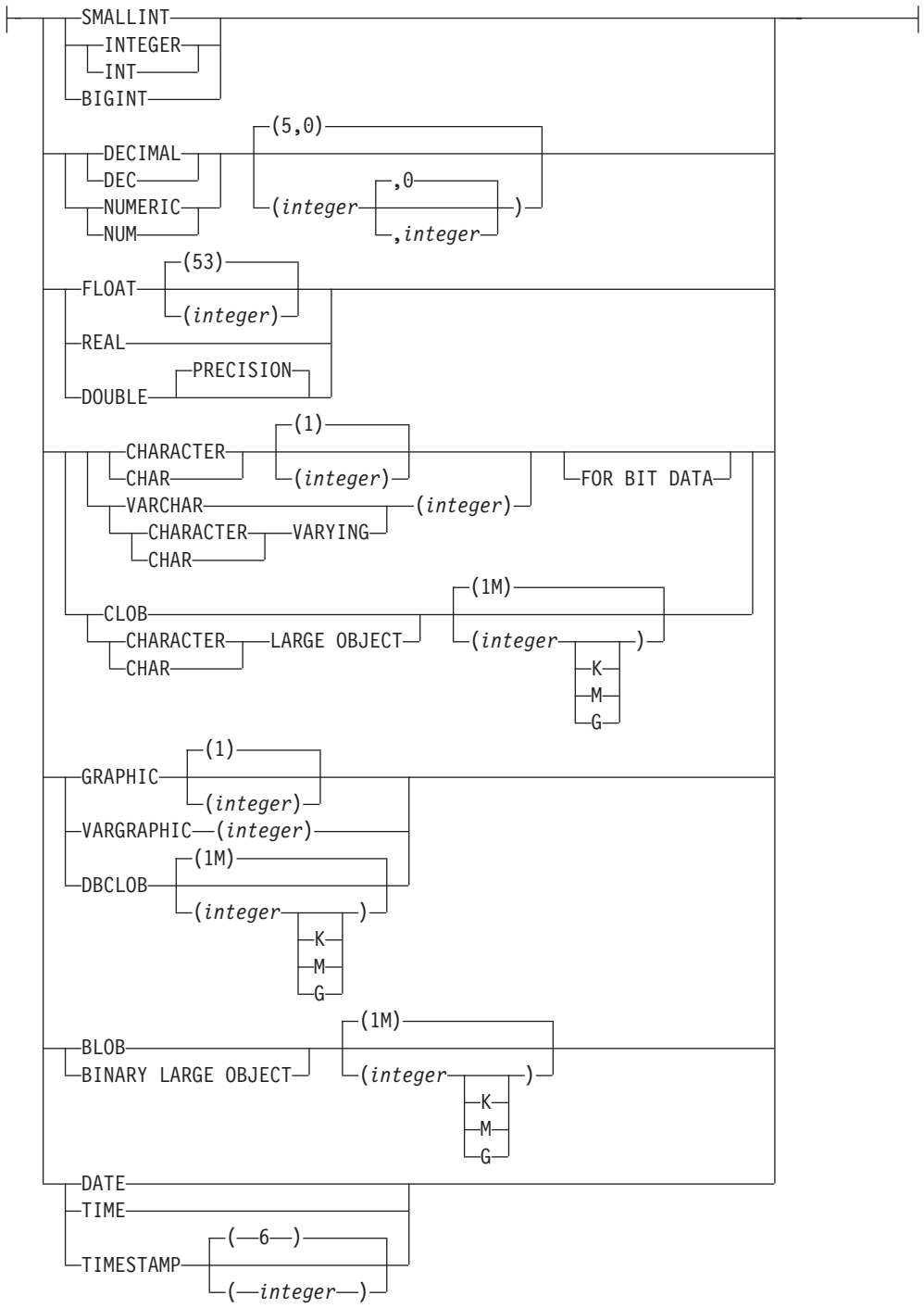
Syntax



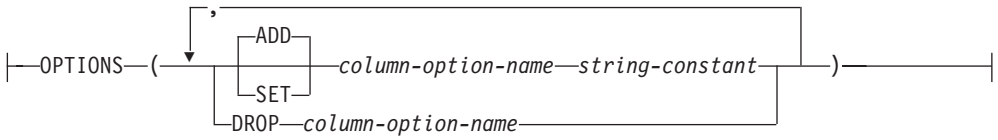


local-data-type:

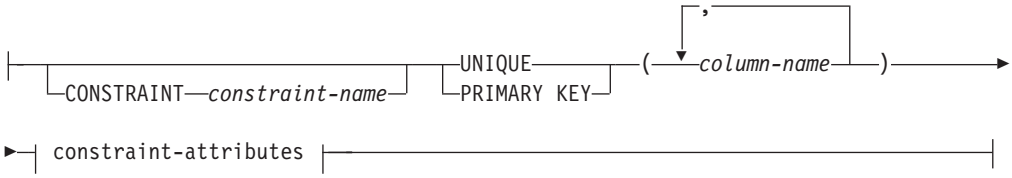
built-in-type:



federated-column-options:

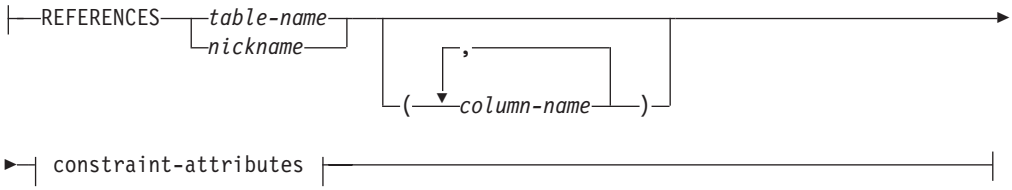


unique-constraint:

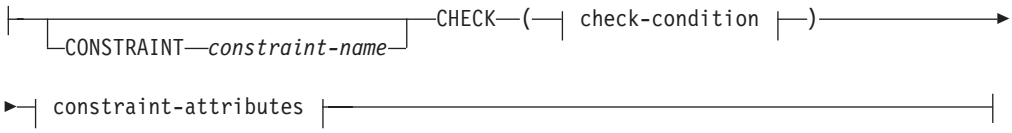


referential-constraint:

references-clause:

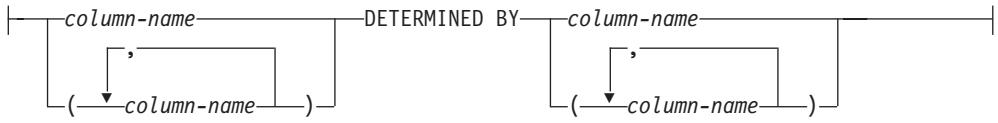


check-constraint:

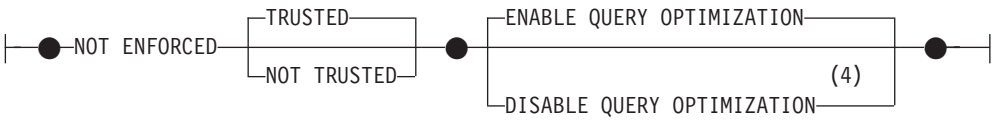


check-condition:

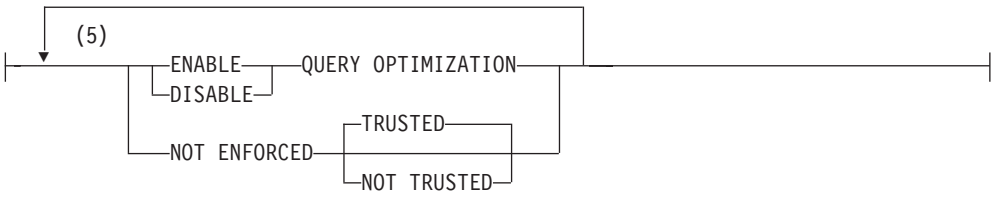
functional-dependency:



constraint-attributes:



constraint-alteration:



Notes:

1 You cannot specify both the ALTER COLUMN clause and an ADD, ALTER, or DROP informational constraint clause in the same ALTER NICKNAME statement.

2 If you need to specify the federated-column-options clause in addition to the LOCAL NAME parameter, the LOCAL TYPE parameter, or both, you must specify the federated-column-options clause last.

3 The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type (SQLSTATE 428H2).

4 DISABLE QUERY OPTIMIZATION is not supported for a unique or primary key constraint.

5 The same clause must not be specified more than once.

Description

nickname
Identifies the nickname for the data source object (such as a table, view, or file) that contains the column being altered. It must be a nickname described in the catalog.

OPTIONS
Indicates the nickname options that are added, set, or dropped when the nickname is altered.

ADD
Adds a nickname option.

SET
Changes the setting of a nickname option.

nickname-option-name
Names a nickname option that is to be added or set.

string-constant
Specifies the setting for *nickname-option-name* as a character string constant.

DROP *nickname-option-name*
Drops a nickname option.

ALTER COLUMN *column-name*
Names the column to be altered. The *column-name* is the federated server's current name for the column of the table or view at the data source. The *column-name* must identify an existing column of the nickname (SQLSTATE 42703). You cannot reference the same column name multiple times in the same ALTER NICKNAME statement (SQLSTATE 42711).

LOCAL NAME *column-name*
Specifies a new name, *column-name*, by which the federated server is to reference the column to be altered. The new name cannot be qualified, and the same name cannot be used for more than one column of the nickname (SQLSTATE 42711).

LOCAL TYPE *local-data-type*
Specifies a new local data type to which the data type of the column that is to be altered will map. The new type is denoted by *local-data-type*.

Some wrappers only support a subset of the SQL data types. For descriptions of specific data types, see the description of the "CREATE TABLE" statement.

OPTIONS
Indicates what column options are to be added, set, or dropped for the column specified after the COLUMN keyword.

ADD
Adds a column option.

SET
Changes the setting of a column option.

column-option-name
Names a column option that is to be added or set.

string-constant
Specifies the setting for *column-option-name* as a character string constant.

DROP *column-option-name*
Drops a column option.

ADD *unique-constraint*
Defines a unique constraint. See the description of the "CREATE NICKNAME" statement.

ADD *referential-constraint*
Defines a referential constraint. See the description of the "CREATE NICKNAME" statement.

ADD *check-constraint*
Defines a check constraint. See the description of the "CREATE NICKNAME" statement.

ALTER FOREIGN KEY *constraint-name*
Alters the constraint attributes of the referential constraint *constraint-name*. For a description of the constraint attributes, see the "CREATE NICKNAME" statement. The *constraint-name* must identify an existing referential constraint (SQLSTATE 42704).

ALTER CHECK *constraint-name*
Alters the constraint attributes of the check constraint *constraint-name*. The *constraint-name* must identify an existing check constraint (SQLSTATE 42704).

constraint-alteration
Provides options for changing the attributes associated with referential or check constraints.

ENABLE QUERY OPTIMIZATION
The constraint can be used for query optimization under appropriate circumstances.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

NOT ENFORCED
Specifies that the constraint is not enforced by the database manager during normal operations such as insert, update, or delete.

TRUSTED
The data can be trusted to conform to the constraint. TRUSTED must be used only if the data in the table is independently known to conform to the constraint. Query results might be unpredictable if the data does not actually conform to the constraint. This is the default option.

NOT TRUSTED
The data cannot be trusted to conform to the constraint. NOT TRUSTED is intended for cases where the data conforms to the constraint for most rows, but it is not independently known that all the rows or future additions will conform to the constraint. If a constraint is NOT TRUSTED and enabled for query optimization, then it will not be used to perform optimizations that depend on the data conforming completely to the constraint. NOT TRUSTED can be specified only for referential integrity constraints (SQLSTATE 42613).

DROP PRIMARY KEY
Drops the definition of the primary key and all referential constraints that are dependent upon this primary key. The nickname must have a primary key.

DROP FOREIGN KEY *constraint-name*
Drops the referential constraint *constraint-name*. The *constraint-name* must identify an existing referential constraint defined on the nickname.

DROP UNIQUE *constraint-name*
Drops the definition of the unique constraint *constraint-name* and all referential constraints that are dependent upon this unique constraint. The *constraint-name* must identify an existing unique constraint.

DROP CHECK *constraint-name*
Drops the check constraint *constraint-name*. The *constraint-name* must identify an existing check constraint defined on the nickname.

DROP CONSTRAINT *constraint-name*
Drops the constraint *constraint-name*. The *constraint-name* must identify an existing check constraint, referential constraint, primary key, or unique constraint defined on the nickname.

ALLOW CACHING or DISALLOW CACHING
Specifies whether the nickname can be referenced in a query that defines a materialized query table, which could be used to cache data from the data source at the federated server.

ALLOW CACHING
Specifies that the nickname can be referenced in a query that defines a materialized query table, which allows data from the data source to be

cached in the materialized query table at the federated server. The refreshable options defined for the materialized query table specify how the cached data in the materialized query table is maintained.

DISALLOW CACHING
Specifies that the nickname cannot be referenced in a query that defines a materialized query table. DISALLOW CACHING cannot be specified for a nickname that is referenced in the fullselect of a materialized query table definition (SQLSTATE 42917).

Rules

- If a nickname is used in a view, SQL method, or SQL function, or informational constraints are defined on it, the ALTER NICKNAME statement cannot be used to change the local names or data types for the columns in the nickname (SQLSTATE 42893). The statement can be used, however, to add, set, or drop column options, nickname options, or informational constraints.
- If a nickname is referenced by a materialized query table definition, the ALTER NICKNAME statement cannot be used to change the local names, data types, column options, or nickname options (SQLSTATE 42893). Moreover, the statement cannot be used to disable caching (SQLSTATE 42917). The statement can be used, however, to add, alter, or drop informational constraints.
- A column option cannot be specified more than once in the same ALTER NICKNAME statement (SQLSTATE 42853). When a column option is enabled, reset, or dropped, any other column options that are in use are not affected.
- For relational nicknames, the ALTER NICKNAME statement within a given unit of work (UOW) cannot be processed under either of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement is already issued in the same UOW against the nickname that is referenced in this statement
- For non-relational nicknames, the ALTER NICKNAME statement within a given unit of work (UOW) cannot be processed under any of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - A nickname referenced in this statement is already referenced by a SELECT statement in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement has already been issued in the same UOW against the nickname that is referenced in this statement

Notes

- If the ALTER NICKNAME statement is used to change the local name for a column of a nickname, queries against that column must reference it by its new name.
- When the local specification of a column's data type is changed, the database manager invalidates any statistics (HIGH2KEY, LOW2KEY, and so on) gathered for that column.
- ***Caching and protected objects:*** For nicknames whose data source object is protected, specify `DISALLOW CACHING`. This ensures that each time the nickname is used, data for the appropriate authorization ID is returned from the data source at query execution time. This is done by restricting the nickname from being

used in the definition of a materialized query table at the federated server, which might be being used to cache the nickname data.

Examples

- *Example 1:* The nickname NICK1 references a DB2 for i table called T1. Also, COL1 is the local name that references this table's first column, C1. Rename the local name for C1 from COL1 to NEWCOL.

```
ALTER NICKNAME NICK1
  ALTER COLUMN COL1
  LOCAL NAME NEWCOL
```

- *Example 2:* The nickname EMPLOYEE references a DB2 for z/OS table called EMP. Also, SALARY is the local name that references EMP_SAL, one of this table's columns. The column's data type, FLOAT, maps to the local data type, DOUBLE. Change the mapping so that FLOAT maps to DECIMAL (10, 5).

```
ALTER NICKNAME EMPLOYEE
  ALTER COLUMN SALARY
  LOCAL TYPE DECIMAL(10,5)
```

- *Example 3:* Indicate that in an Oracle table, a column with the data type of VARCHAR does not have trailing blanks. The nickname for the table is NICK2, and the local name for the column is COL1.

```
ALTER NICKNAME NICK2
  ALTER COLUMN COL1
  OPTIONS (ADD VARCHAR_NO_TRAILING_BLANKS 'Y')
```

- *Example 4:* Alter the fully qualified path for the table-structured file, drugdata1.txt, for the nickname DRUGDATA1. Use the FILE_PATH nickname option and change the path from the current value of '/user/pat/drugdata1.txt' to '/usr/kelly/data/drugdata1.txt'.

```
ALTER NICKNAME DRUGDATA1
  OPTIONS (SET FILE_PATH '/usr/kelly/data/drugdata1.txt')
```

ALTER PACKAGE

The ALTER PACKAGE statement alters bind options for a package at the current server without having to bind or rebind the package.

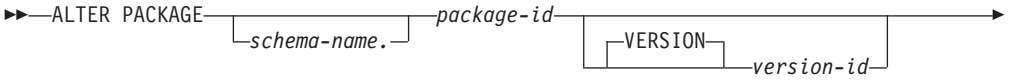
Invocation

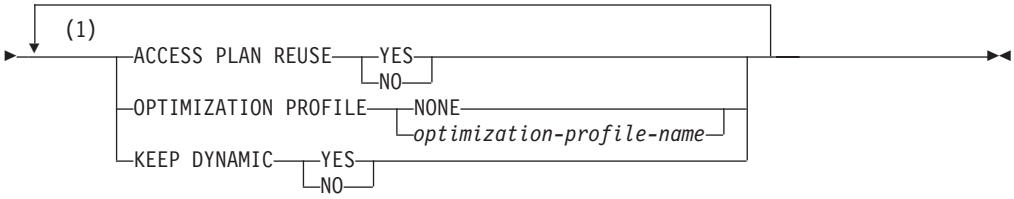
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTERIN privilege on the schema
- BIND privilege on the package
- DBADM authority





Notes:

1 The same clause must not be specified more than once.

Description

schema-name.package-id
Identifies the package that is to be altered. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The schema name and package ID, together with the implicitly or explicitly specified version ID, must identify a package that exists at the current server (SQLSTATE 42704).

VERSION *version-id*
Identifies which package version is to be altered. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be altered in one invocation of the ALTER PACKAGE statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

ACCESS PLAN REUSE
Indicates whether the query compiler should attempt to reuse the access plans for static statements in the package during future implicit and explicit rebinds.

NO Specifies not to reuse access plans.

YES
Specifies to attempt to reuse access plans.

OPTIMIZATION PROFILE
Indicates what, if any, optimization profile to associate with the package.

NONE
Associates no optimization profile with the package. If an optimization profile is already associated with the package, the association is removed.

optimization-profile-name
Associates the optimization profile *optimization-profile-name* with the package. The optimization profile is a two-part name. If the specified *optimization-profile-name* is unqualified, the value of the CURRENT DEFAULT SCHEMA special register is used as the implicit qualifier. If an optimization profile is already associated with the package, the association is replaced with *optimization-profile-name*.

While the ALTER PACKAGE statement removes the current copy of the package from the DB2 package cache, it does not invalidate the package and does not cause an implicit rebind to take place. This means that although dynamic SQL is affected by the changes made by the statement, query execution plans for static statements are not be affected until the next implicit or explicit rebind.

KEEP DYNAMIC

Starting with DB2 for Linux, UNIX, and Windows Version 9.8 Fix Pack 2, you can modify the value of the `KEEPDYNAMIC` bind option for a package without requiring a fresh bind operation, thereby avoiding unnecessary recompilation until the next bind operation occurs. This option controls how long the statement text and section associated with a prepared statement are kept in the SQL context. It takes effect after all applications that are using the package have completed the transactions that were running when the **ALTER PACKAGE** statement was executed.

YES

Instructs the SQL context to keep the statement text and section associated with prepared statements indefinitely. Dynamic SQL statements are kept across transactions. All packages bound with `KEEPDYNAMIC YES` are by default compatible with the existing package cache behavior.

NO

Instructs the SQL context to remove the statement text and section associated with prepared statements at the end of each unit of work. The executable versions of prepared statements and the statement text in packages bound with the `KEEP DYNAMIC NO` option are removed from the SQL context at transaction boundaries. The client, driver, or application needs to prepare any dynamic SQL statement it wishes to reuse in a new unit of work again.

For remote applications that use an IBM non-embedded API, once you have ensured that statements will be prepared in new transactions, you can use this option so that WLB will not be disallowed solely based on the KEEP DYNAMIC behavior. However even with this option, WLB may be disallowed for other reasons.

SELECT statements issued by cursors with the WITH HOLD option are disassociated from the SQL context at the next transaction boundary where the cursor is closed. As a result, workload balancing is allowed as long as there are no executable versions of prepared statements associated with the application in the SQL context.

Note: Workload balancing is not restricted for dynamic SQL applications that use IBM non-embedded APIs, such as JDBC, .NET, or CLI/ODBC, to run SQL within the common client packages. These interfaces implicitly re-prepare SQL statements before executing them in transactions where their connection might have been moved to a new executable version of prepared statements.

Notes

- **Catalog view values may not reflect the settings that were in effect for the package:** Because this statement does not trigger a rebind of the package, the settings for a package as shown in the SYSCAT.PACKAGES catalog view might not reflect what was actually in effect during the last BIND or REBIND. If the ALTER_TIME is greater than the LAST_BIND_TIME, then this might be the case.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with the BIND and REBIND commands. These alternatives are non-standard and should not be used.
  - APREUSE can be specified in place of ACCESS PLAN REUSE.
  - OPTPROFILE can be specified in place of OPTIMIZATION PROFILE.
  - KEEPDYNAMIC can be specified in place of KEEP DYNAMIC.

Examples

Example 1: Enable access plan reuse for package TRUUVERT.EMPADMIN.

```
ALTER PACKAGE TRUUVERT.EMPADMIN ACCESS PLAN REUSE YES
```

Example 2: Assume access plan reuse has been enabled for package TRUUVERT.EMPADMIN. Assume also that optimization profile AYYANG.INDEXHINTS contains a statement profile for a specific statement within the package. Associate the optimization profile with this package so that it will override the reuse of the access plan for the statement.

```
ALTER PACKAGE TRUUVERT.EMPADMIN OPTIMIZATION PROFILE AYYANG.INDEXHINTS
```

Dynamic statements will be affected after the statement commits; static statements will be affected at the next rebind. When the package is rebound, the query compiler will attempt to reuse the access plans for all static statements in the package, with the exception of the statement identified by the optimization profile. When recompiling this statement, the query compiler will instead attempt to apply the statement profile.

Example 3: The following statement will result in no optimization profile being associated with package TRUUVERT.EMPADMIN.

```
ALTER PACKAGE TRUUVERT.EMPADMIN OPTIMIZATION PROFILE NONE
```

ALTER PERMISSION

The ALTER PERMISSION statement alters a row permission that exists at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

Description

permission-name
This is the name of the row permission to be altered. The name must identify a row permission that already exists at the current server (SQLSTATE 42704). The name must not identify a default row permission that is created implicitly by the database manager (SQLSTATE 428H9).

ENABLE

Enables the row permission. If row level access control is not currently activated on the table, the row permission will become effective when row level access control is activated on the table. If row level access control is currently activated on the table, the row permission becomes effective immediately and all packages and dynamic cached statements that reference the table are invalidated.

ENABLE is ignored if the row permission is already defined as enabled.

DISABLE

Disables the row permission. If row level access control is not currently activated on the table, the row permission will remain ineffective when row level access control is activated on the table. If row level access control is currently activated on the table, the row permission becomes ineffective immediately and all packages and dynamic cached statements that reference the table are invalidated.

DISABLE is ignored if the row permission is already defined as disabled.

Examples

- *Example 1*: Enable permission P1.

```
ALTER PERMISSION P1 ENABLE
```

- *Example 2*: Disable permission P1.

```
ALTER PERMISSION P1 DISABLE
```

ALTER PROCEDURE (external)

The ALTER PROCEDURE (External) statement modifies an existing external procedure by changing the properties of the procedure.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTERIN privilege on the schema of the procedure
- Owner of the procedure, as recorded in the OWNER column of the SYSCAT.ROUTINES catalog view
- DBADM authority

To alter the EXTERNAL NAME of a procedure, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

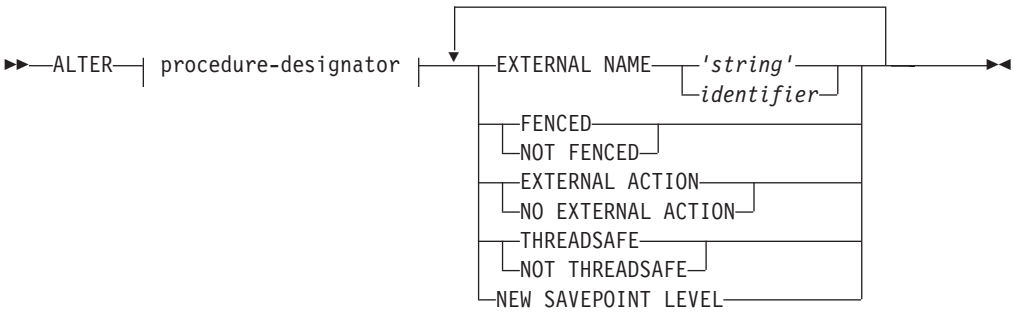
- CREATE_EXTERNAL_ROUTINE authority on the database
- DBADM authority

To alter a procedure to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

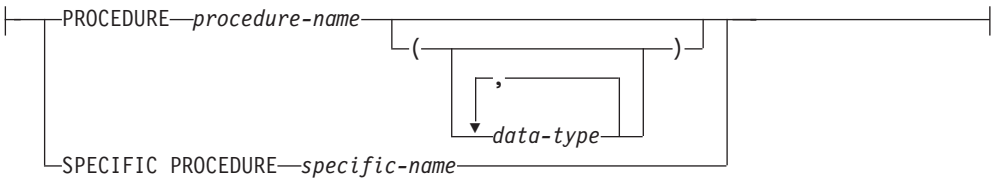
- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

To alter a procedure to be fenced, no additional authorities or privileges are required.

Syntax



procedure-designator:



Description

procedure-designator
Identifies the procedure to alter. The *procedure-designator* must identify a procedure that exists at the current server. The owner of the procedure and all privileges on the procedure are preserved. For more information, see "Function, method, and procedure designators" on page 20.

EXTERNAL NAME *'string'* **or** *identifier*
Identifies the name of the user-written code that implements the procedure.

FENCED or NOT FENCED
Specifies whether the procedure is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most procedures have the option of running as FENCED or NOT FENCED.

If a procedure is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the procedure. In general, a procedure running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for procedures that were not adequately coded, reviewed, and tested can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED stored procedures are used.

A procedure declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a procedure has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the procedure cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE or CLR procedures (SQLSTATE 42849).

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not (NO EXTERNAL ACTION). If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

THREADSAFE or NOT THREADSAFE
Specifies whether the procedure is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the procedure is defined with LANGUAGE other than OLE:

- If the procedure is defined as THREADSAFE, the database manager can invoke the procedure in the same process as other routines. In general, to be

threadsafe, a procedure should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED procedures can be THREADSAFE.

- If the procedure is defined as NOT THREADSAFE, the database manager will never invoke the procedure in the same process as another routine. Only a fenced procedure can be NOT THREADSAFE (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE procedures (SQLSTATE 42849).

NEW SAVEPOINT LEVEL
Specifies that a new savepoint level is to be created for the procedure. A savepoint level refers to the scope of reference for any savepoint-related statement, as well as to the name space used for comparison and reference of any savepoint names.

The savepoint level for a procedure can only be altered to NEW SAVEPOINT LEVEL.

Rules

- It is not possible to alter a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

Example

Alter the procedure PARTS_ON_HAND() to be not fenced.

```
ALTER PROCEDURE PARTS_ON_HAND() NOT FENCED
```

ALTER PROCEDURE (sourced)

The ALTER PROCEDURE (Sourced) statement modifies an existing sourced procedure by changing the data type of one or more parameters of the sourced procedure.

Invocation

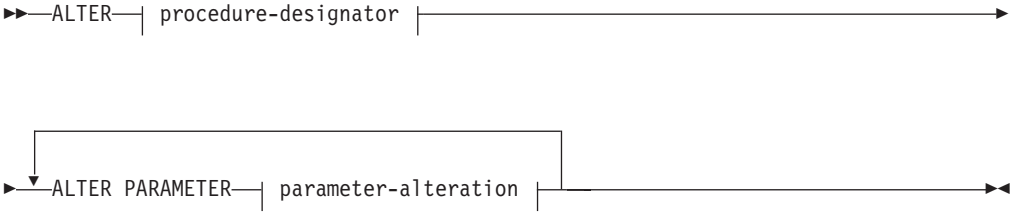
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

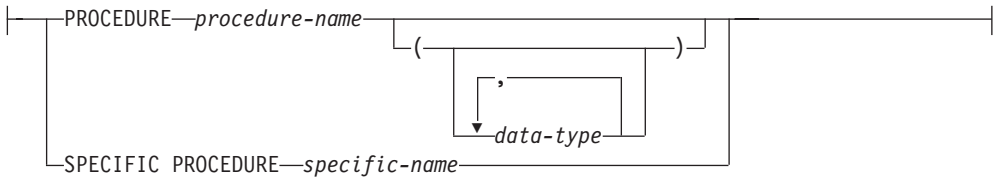
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTERIN privilege on the schema of the procedure
- Owner of the procedure, as recorded in the OWNER column of the SYSCAT.ROUTINES catalog view
- DBADM authority

Syntax



procedure-designator:



parameter-alteration:

Description

procedure-designator
Uniquely identifies the procedure to be altered. The identified procedure must be a sourced procedure (SQLSTATE 42849). For more information, see "Function, method, and procedure designators" on page 20.

parameter-name
Identifies the parameter to be altered. The *parameter-name* must identify an

existing parameter of the procedure (SQLSTATE 42703). The name must not identify a parameter that is otherwise being altered in the same ALTER PROCEDURE statement (SQLSTATE 42713).

data-type
Specifies the new local data type of the parameter. SQL data type specifications and abbreviations that are valid for the *data-type* definition of a CREATE TABLE statement can be specified. BLOB, CLOB, DBCLOB, DECFLOAT, XML, REFERENCE, and user-defined types are not supported (SQLSTATE 42815).

Example

Assume that federated procedure FEDEMPLOYEE has been created for a remote Oracle procedure named 'EMPLOYEE'. The data type of an input parameter named SALARY maps to a DOUBLE(8) in DB2. Alter the data type of this parameter to DECIMAL(5,2).

```
ALTER PROCEDURE FEDEMPLOYEE
  ALTER PARAMETER SALARY
  SET DATA TYPE DECIMAL(5,2)
```

ALTER PROCEDURE (SQL)

The ALTER PROCEDURE (SQL) statement modifies an existing SQL procedure by changing the properties of the procedure.

Invocation

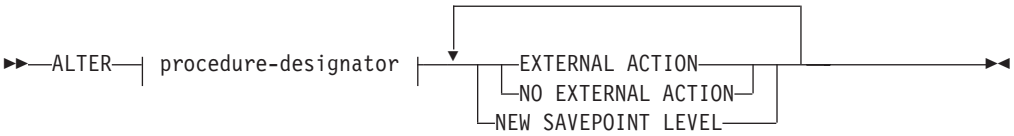
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

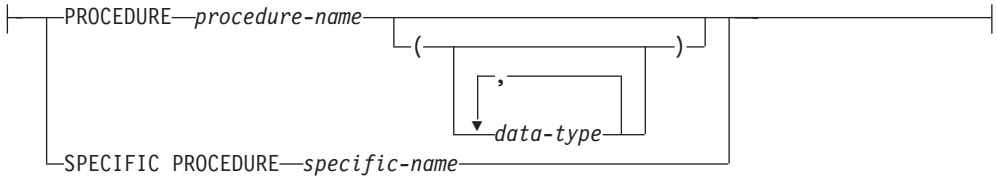
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTERIN privilege on the schema of the procedure
- Owner of the procedure, as recorded in the OWNER column of the SYSCAT.ROUTINES catalog view
- DBADM authority

Syntax



procedure-designator:



Description

procedure-designator
Identifies the procedure to alter. The *procedure-designator* must identify a procedure that exists at the current server. The owner of the procedure and all privileges on the procedure are preserved. For more information, see "Function, method, and procedure designators" on page 20.

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not (NO EXTERNAL ACTION). If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

NEW SAVEPOINT LEVEL
Specifies that a new savepoint level is to be created for the procedure. A

savepoint level refers to the scope of reference for any savepoint-related statement, as well as to the name space used for comparison and reference of any savepoint names.

The savepoint level for a procedure can only be altered to NEW SAVEPOINT LEVEL.

Rules

- It is not possible to alter a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

Example

Alter the procedure MEDIAN_RESULT_SET to indicate that it has no external action.

```
ALTER PROCEDURE MEDIAN_RESULT_SET(DOUBLE)
  NO EXTERNAL ACTION
```

ALTER SCHEMA

The ALTER SCHEMA statement modifies the data capture attribute of an existing schema.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- Owner of the schema, as recorded in the OWNER column of SYSCAT.SCHEMATA catalog view
- DBADM authority

Syntax

Description

schema-name
Identifies the schema to be altered. The *schema-name* must identify a schema that exists at the current server (SQLSTATE 42704).

DATA CAPTURE
Indicates whether extra information for data replication is to be written to the log.

NONE
Indicates that no extra information for data replication will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this schema will be written to the log. This option is required if this schema will be replicated and a replication capture program is used to capture changes for this schema from the log.

Notes

- Altering the DATA CAPTURE attribute at the schema level causes newly created tables to inherit the DATA CAPTURE attribute from the schema if one is not specified at the table level. Altering the DATA CAPTURE attribute at the schema level does not affect the DATA CAPTURE attribute of existing tables within that schema. If the DATA CAPTURE attribute is changed and any existing tables do not match the new attribute, a warning is returned (SQLSTATE 01696).
- To find the list of tables that have the DATA CAPTURE attribute set to CHANGES, issue the following query:

```
SELECT TABNAME, TABSCHEMA FROM SYSCAT.TABLES
  WHERE TYPE IN ('T','S','L')
    AND DATACAPTURE <> 'N'
```

- To find the list of tables that have the DATA CAPTURE attribute set to NONE, issue the following query:

```
SELECT TABNAME, TABSCHEMA FROM SYSCAT.TABLES
  WHERE TYPE IN ('T','S','L')
    AND DATACAPTURE = 'N'
```

ALTER SECURITY LABEL COMPONENT

The ALTER SECURITY LABEL COMPONENT statement modifies a security label component.

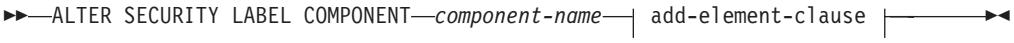
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

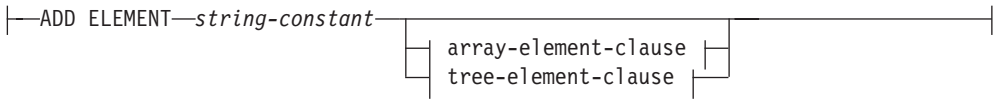
Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax

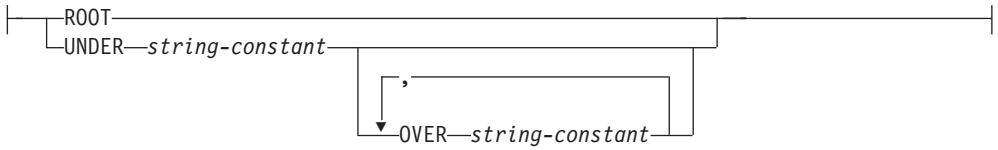


add-element-clause:



array-element-clause:

tree-element-clause:



Description

component-name
Specifies the name of the security label component to be altered. The named component must exist at the current server (SQLSTATE 42704).

ADD ELEMENT
Specifies the element to be added to the security label component. If *array-element-clause* and *tree-element-clause* are not specified, the element is added to a set component.

string-constant
The string constant value to be added to the set of valid values for the

security label component. The value cannot be the same as any other value in the set of valid values for the security label component (SQLSTATE 42713).

BEFORE or AFTER
For an array component, specifies where the element is to be added in the ordered set of element values for the security label component.

BEFORE
The element to be added is to be ranked immediately before the identified existing element.

AFTER
The element to be added is to be ranked immediately after the identified existing element.

string-constant
Specifies a string constant value of an existing element in the array component (SQLSTATE 42704).

ROOT or UNDER
For a tree component, specifies where the element is to be added in the tree structure of node element values for the security label component.

ROOT
The element to be added is to be considered the root node of the tree.

UNDER *string-constant*
The element to be added is an immediate child of the element identified by the *string-constant*. The *string-constant* value must be an existing element in the tree component (SQLSTATE 42704).

OVER *string-constant*,...
The element to be added is an immediate child of every element identified by the list of *string-constant* values. Each *string-constant* value must be an existing element in the tree component (SQLSTATE 42704).

Rules

- Element names cannot contain any of these characters (SQLSTATE 42601):
  - Opening parenthesis - (
  - Closing parenthesis -)
  - Comma - ,
  - Colon - :
- An element name can have no more than 32 bytes (SQLSTATE 42622).
- If a security label component is a set or a tree, no more than 64 elements can be part of that component.
- If the component is an array, it might or might not be possible to arrive at an array whose total number of elements matches the total number of elements that could be specified when creating a security label component of type array (65 535). DB2 assigns an encoded value to the new element from within the interval into which the new element is added. Depending on the pattern followed when adding elements to an array component, the number of possible values that can be assigned from within a particular interval might be quickly exhausted if several elements are inserted into that interval.
- BEFORE and AFTER must only be specified for a security label component that is an array (SQLSTATE 42613).

- ROOT and UNDER must only be specified for a security label component that is a tree (SQLSTATE 42613).

Notes

- For a set component, there is no order to the elements in the set.

Examples

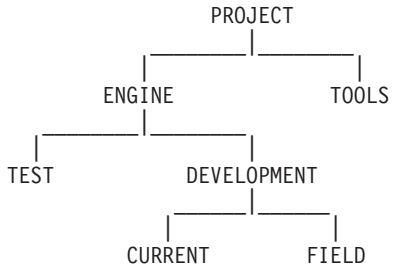
- *Example 1:* Add the element 'High classified' to the LEVEL security label array component between the elements 'Secret' and 'Classified'.

```
ALTER SECURITY LABEL COMPONENT LEVEL
  ADD ELEMENT 'High classified' BEFORE 'Classified'
```

- *Example 2:* Add the element 'Funding' to the COMPARTMENTS security label set component.

```
ALTER SECURITY LABEL COMPONENT COMPARTMENTS
  ADD ELEMENT 'Funding'
```

- *Example 3:* Add the elements 'ENGINE' and 'TOOLS' to the GROUPS security label array component. The following diagram shows where these new elements are to be placed.




```
ALTER SECURITY LABEL COMPONENT GROUPS
  ADD ELEMENT 'TOOLS' UNDER 'PROJECT'

ALTER SECURITY LABEL COMPONENT GROUPS
  ADD ELEMENT 'ENGINE' UNDER 'PROJECT'
    OVER 'TEST', 'DEVELOPMENT'
```

ALTER SECURITY POLICY

The ALTER SECURITY POLICY statement modifies a security policy.

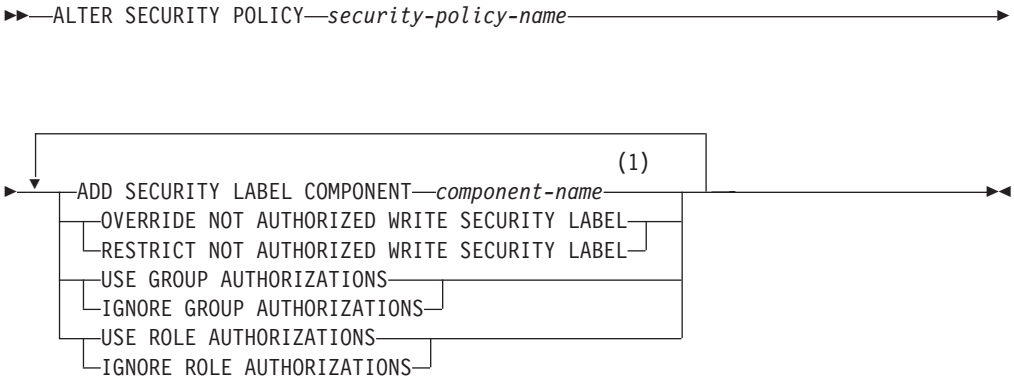
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Notes:

1 Only the ADD SECURITY LABEL COMPONENT clause can be specified more than once.

Description

security-policy-name
Specifies the name of the security policy to be altered. The name must identify an existing security policy at the current server (SQLSTATE 42710).

ADD SECURITY LABEL COMPONENT *component-name*
Adds a security label component to the security policy. The same security component must not be specified more than once for the security policy (SQLSTATE 42713). The security policy cannot currently be in use by a table (SQLSTATE 42893).

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL or RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
Specifies the action taken when a user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement issued against a table that is protected with this security policy. A user's security label and exemption credentials determine the user's authorization to write an explicitly provided security label.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL
Indicates that the value of the user's security label, rather than the explicitly specified security label, is used for write access during an insert or update operation.

RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
Indicates that the insert or update operation will fail if the user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement (SQLSTATE 42519).

USE GROUP AUTHORIZATION or IGNORE GROUP AUTHORIZATION
Specifies whether or not security labels and exemptions granted to groups, directly or indirectly, are considered for any access attempt.

USE GROUP AUTHORIZATION
Indicates that any security labels or exemptions granted to groups, directly or indirectly, are considered.

IGNORE GROUP AUTHORIZATION
Indicates that any security labels or exemptions granted to groups are not considered.

USE ROLE AUTHORIZATION or IGNORE ROLE AUTHORIZATION
Specifies whether or not security labels and exemptions granted to roles, directly or indirectly, are considered for any access attempt.

USE ROLE AUTHORIZATION
Indicates that any security labels or exemptions granted to roles, directly or indirectly, are considered.

IGNORE ROLE AUTHORIZATION
Indicates that any security labels or exemptions granted to roles are not considered.

Rules

- If a user does not directly hold a security label for write access, an error is returned in the following situations (SQLSTATE 42519):
  - A value for the row security label column is not explicitly provided as part of the SQL statement
  - The OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option is in effect for the security policy, and the user is not allowed to write a data object with the provided security label

Notes

- New components are logically added at the end of the existing security label definition contained by the modified policy. Existing security labels defined for this security policy are modified to contain the new component as part of their definition with no element in their value for this component.
- ***Cache invalidation when changing NOT AUTHORIZED WRITE SECURITY LABEL:*** Changing the NOT AUTHORIZED WRITE SECURITY LABEL to a new value will cause the invalidation of any cached dynamic or static SQL statements that are dependent on any table that is protected by the security policy being altered.
- Because the session authorization ID is the focus authorization ID for label-based access control, security labels granted to groups or to roles that are accessible through groups are eligible for consideration for all types of SQL statements, including static SQL.

- If more than one security label or exemption is available to a user with associated groups or roles at the time of a read or write access attempt, those security labels and exemptions will be evaluated for eligibility based on the following rules:
  - If the security policy enables only role authorizations for consideration, all security labels and exemptions granted to roles of which the user authorization ID is a direct or indirect member will be considered. Security labels and exemptions granted to roles for which membership is only accessible through the groups associated with the user authorization ID will not be considered.
  - If the security policy enables only group authorizations for consideration, all security labels and exemptions granted to groups associated with the user authorization ID will be considered. Security labels and exemptions granted to roles for which membership is only accessible through the groups associated with the user authorization ID will not be considered.
  - If the security policy enables both group and role authorizations for consideration, any security labels and exemptions granted to roles accessible to the user indirectly through groups associated with the user authorization ID will be considered.
  - Role authorizations that are accessible to the user only through PUBLIC will not be considered at any time.
- If more than one security label is eligible for consideration during an access attempt, the values provided for each security label are merged at the individual component level to form a security label that reflects the combination of all available values at each component piece of the security policy. This is the security label value that will be used for the access attempt.

  The mechanisms for combining security labels vary by component type. The components of the resultant security label are as follows:
  - Set components contain the union of all unique values encountered in the eligible security labels
  - Array components contain the highest order element encountered in the eligible security labels
  - Tree components contain the union of all unique values encountered in the eligible security labels
- If more than one exemption is eligible for consideration during an access attempt, all found exemptions are applied to the access attempt.

Examples

- *Example 1:* Alter a security policy named DATA_ACCESS to add a new component named REGION.

```
ALTER SECURITY POLICY DATA_ACCESS
  ADD COMPONENT REGION
```

- *Example 2:* Alter a security policy named DATA_ACCESS to allow access through security labels granted to roles.

```
ALTER SECURITY POLICY DATA_ACCESS
  USE ROLE AUTHORIZATIONS
```

- *Example 3:* Show the eligible security labels that would be considered depending on the settings for group or role authorizations in a security policy. The security policy SECUR_POL has an array component and a set component, consisting of the following elements:

  Array = {TS, S, C, U}

  Set = {A, B, X, Y}

The following security labels are defined for SECUR_POL:

Security label L1 = C:A

Security label L2 = S:B

Security label L3 = TS:X

Security label L4 = U:Y

User Paul is a member of role R1 and group G1. Group G1 is a member of role R2. Security label L1 is granted to Paul. Security label L2 is granted to role R1. Security label L3 is granted to group G1. Security label L4 is granted to role R2. The following table shows what security labels would be considered for any access attempt by Paul, depending on the different possible settings of the security policy SECUR_POL.

Table 11. Security labels considered as a function of security policy settings

	Roles Enabled	**Roles Disabled**
Groups Enabled	L1, L2, L3, L4	L1, L3
Groups Disabled	L1, L2	L1

The following table shows the value of the combined security label for any access attempt by Paul, depending on the different settings of the security policy SECUR_POL.

Table 12. Combined security labels as a function of security policy settings

	Roles Enabled	**Roles Disabled**
Groups Enabled	TS:(A, B, X, Y)	TS:(A, X)
Groups Disabled	S:(A, B)	C:A

ALTER SEQUENCE

The ALTER SEQUENCE statement can be used to change a sequence.

A sequence can be changed in the following ways:

- Restarting the sequence
- Changing the increment between future sequence values
- Setting or eliminating the minimum or maximum values
- Changing the number of cached sequence numbers
- Changing the attribute that determines whether the sequence can cycle or not
- Changing whether sequence numbers must be generated in order of request

Invocation

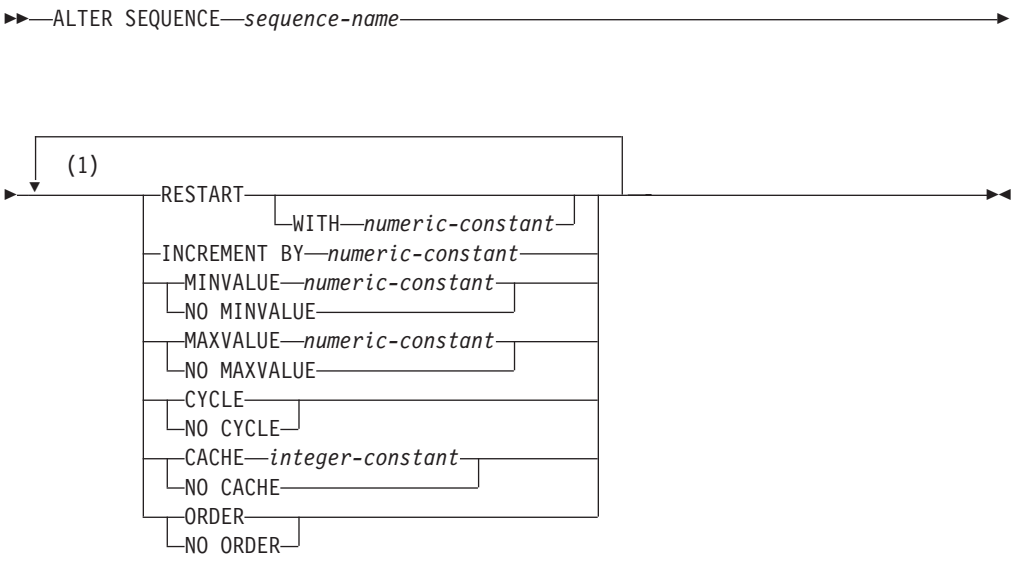
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTER privilege on the sequence to be altered
- ALTERIN privilege on the schema implicitly or explicitly specified
- DBADM authority

Syntax



Notes:

1 The same clause must not be specified more than once.

Description

sequence-name
Identifies the sequence that is to be changed. The name, including the implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists in the explicitly or implicitly specified schema, an error (SQLSTATE 42704) is returned. *sequence-name* must not be a sequence generated by the system for an identity column (SQLSTATE 428FB).

RESTART
Restarts the sequence. If *numeric-constant* is not specified, the sequence is restarted at the value specified implicitly or explicitly as the starting value on the CREATE SEQUENCE statement that originally created the sequence.

WITH *numeric-constant*
Restarts the sequence with the specified value. This value can be any positive or negative value that could be assigned to a column of the data type associated with the sequence (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA).

INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the sequence. This value can be any positive or negative value that could be assigned to a column of the data type associated with the sequence (SQLSTATE 42815). The value must not exceed the value of a large integer constant (SQLSTATE 42820) and must not contain nonzero digits to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, then this is a descending sequence. If this value is 0 or positive, this is an ascending sequence after the ALTER statement.

MINVALUE or NO MINVALUE
Specifies the minimum value at which a descending sequence either cycles or stops generating values, or an ascending sequence cycles to after reaching the maximum value.

MINVALUE *numeric-constant*
Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to a column of the data type associated with the sequence (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

NO MINVALUE
For an ascending sequence, the value is the original starting value. For a descending sequence, the value is the minimum value of the data type associated with the sequence.

MAXVALUE or NO MAXVALUE
Specifies the maximum value at which an ascending sequence either cycles or stops generating values, or a descending sequence cycles to after reaching the minimum value.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to a column of the data type associated with the sequence (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

NO MAXVALUE
For an ascending sequence, the value is the maximum value of the data type associated with the sequence. For a descending sequence, the value is the original starting value.

CYCLE or NO CYCLE
Specifies whether the sequence should continue to generate values after reaching either its maximum or minimum value. The boundary of the sequence can be reached either with the next value landing exactly on the boundary condition, or by overshooting the value.

CYCLE
Specifies that values continue to be generated for this sequence after the maximum or minimum value has been reached. If this option is used, after an ascending sequence reaches its maximum value, it generates its minimum value; or after a descending sequence reaches its minimum value, it generates its maximum value. The maximum and minimum values for the sequence determine the range that is used for cycling.

When CYCLE is in effect, then duplicate values can be generated by DB2 for the sequence.

NO CYCLE
Specifies that values will not be generated for the sequence once the maximum or minimum value for the sequence has been reached.

CACHE or NO CACHE
Specifies whether to keep some preallocated values in memory for faster access. This is a performance and tuning option.

CACHE *integer-constant*
Specifies the maximum number of sequence values that are preallocated and kept in memory. Preallocating and storing values in the cache reduces synchronous I/O to the log when values are generated for the sequence.

In the event of a system failure, all cached sequence values that have not been used in committed statements are lost (that is, they will never be used). The value specified for the CACHE option is the maximum number of sequence values that could be lost in case of system failure.

The minimum value is 2 (SQLSTATE 42815).

NO CACHE
Specifies that values of the sequence are not to be preallocated. It ensures that there is not a loss of values in the case of a system failure, shutdown or database deactivation. When this option is specified, the values of the sequence are not stored in the cache. In this case, every request for a new value for the sequence results in synchronous I/O to the log.

ORDER or NO ORDER
Specifies whether the sequence numbers must be generated in order of request.

ORDER
Specifies that the sequence numbers are generated in order of request.

NO ORDER
Specifies that the sequence numbers do not need to be generated in order of request.

Notes

- Only future sequence numbers are affected by the ALTER SEQUENCE statement.

- The data type of a sequence cannot be changed. Instead, drop and re-create the sequence specifying the required data type for the new sequence.
- All cached values are lost when a sequence is altered.
- After restarting a sequence or changing to CYCLE, it is possible for sequence numbers to be duplicate values of ones generated by the sequence previously.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - A comma can be used to separate multiple sequence options.
  - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER can be specified in place of NO MINVALUE, NO MAXVALUE, NO CYCLE, NO CACHE, and NO ORDER, respectively

Example

A possible reason for specifying RESTART without a numeric value would be to reset the sequence to the START WITH value. In this example, the goal is to generate the numbers from 1 up to the number of rows in the table and then inserting the numbers into a column added to the table using temporary tables. Another use would be to get results back where all the resulting rows are numbered:

```
ALTER SEQUENCE ORG_SEQ RESTART
SELECT NEXT VALUE FOR ORG_SEQ, ORG.* FROM ORG
```

ALTER SERVER

The ALTER SERVER statement is used to modify the definition or configuration of a data source.

This statement can be used to make the following changes:

- Modify the definition of a specific data source, or the definition of a category of data sources.
- Make changes in the configuration of a specific data source, or the configuration of a category of data sources-changes that will persist over multiple connections to the federated database.

In this statement, the word SERVER and the parameter names that start with *server-* refer only to data sources in a federated system. They do not refer to the federated server in such a system, or to DRDA® application servers.

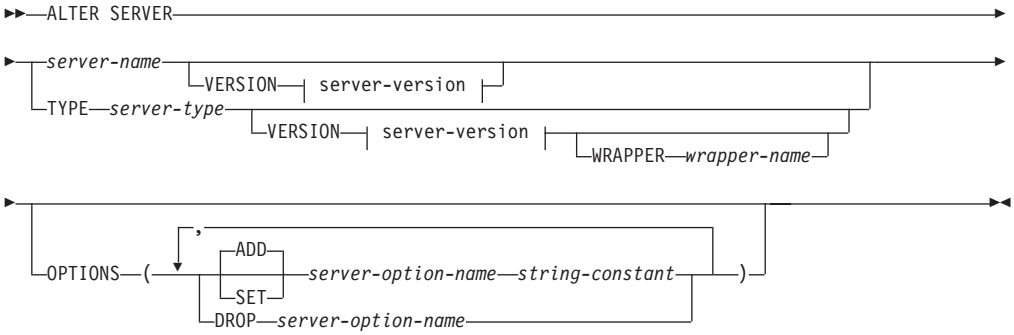
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

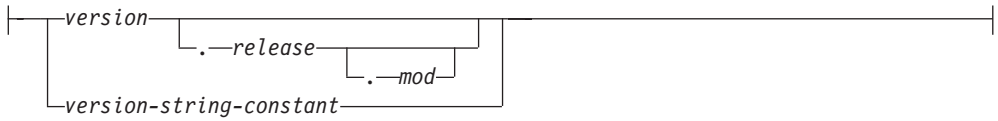
Authorization

The privileges held by the authorization ID of the statement must include DBADM authority.

Syntax



server-version:



Description

server-name
Identifies the federated server's name for the data source to which the changes being requested are to apply. The data source must be one that is described in the catalog.

VERSION
After *server-name*, VERSION and its parameter specify a new version of the data source that *server-name* denotes.

version
Specifies the version number. The value must be an integer.

release
Specifies the number of the release of the version denoted by *version.* The value must be an integer.

mod
Specifies the number of the modification of the release denoted by *release.* The value must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release* and, if applicable, *mod* (for example, '8.0.3').

TYPE *server-type*
Specifies the type of data source to which the changes being requested are to apply.

VERSION
After *server-type*, VERSION and its parameter specify the version of the data sources for which server options are to be enabled, reset, or dropped.

WRAPPER *wrapper-name*
Specifies the name of the wrapper that the federated server uses to interact with data sources of the type and version denoted by *server-type* and *server-version*. The wrapper must be listed in the catalog.

OPTIONS
Indicates what server options are to be enabled, reset, or dropped for the data source denoted by *server-name*, or for the category of data sources denoted by *server-type* and its associated parameters.

ADD
Enables a server option.

SET
Changes the setting of a server option.

server-option-name
Names a server option that is to be enabled or reset.

string-constant
Specifies the setting for *server-option-name* as a character string constant.

DROP *server-option-name*
Drops a server option.

Notes

- A server option cannot be specified more than once in the same ALTER SERVER statement (SQLSTATE 42853). When a server option is enabled, reset, or dropped, any other server options that are in use are not affected.
- An ALTER SERVER statement within a given unit of work (UOW) cannot be processed (SQLSTATE 55007) under either of the following conditions:
  - The statement references a single data source, and the UOW already includes one of the following:

- A SELECT statement that references a nickname for a table or view within this data source
- An open cursor on a nickname for a table or view within this data source
- Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within this data source

– The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following:
- A SELECT statement that references a nickname for a table or view within one of these data sources
- An open cursor on a nickname for a table or view within one of these data sources
- Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within one of these data sources

- If the server option is set to one value for a type of data source, and set to another value for an instance of this type, the second value overrides the first one for the instance. For example, assume that PLAN_HINTS is set to 'Y' for server type ORACLE, and to 'N' for an Oracle data source named DELPHI. This configuration causes plan hints to be enabled at all Oracle data sources except DELPHI.
- You can only alter set or alter drop server options for a category of data sources that was enabled by a prior alter add server option operation (SQLSTATE 42704).
- When altering the server version, DB2 does not verify that the specified server version matches the remote server version. Specifying an incorrect server version can result in SQL errors when you access nicknames that belong to the DB2 server definition. This is most likely when you specify a server version that is later than the remote server version. In that case, when you access nicknames that belong to the server definition, DB2 might send SQL that the remote server does not recognize.

Examples

- *Example 1:* Ensure that when authorization IDs are sent to your Oracle 8.0.3 data sources, the case of the IDs will remain unchanged. Also, assume that the local federated server CPU is twice as fast as the data source CPU. Inform the optimizer of this statistic.

```
ALTER SERVER
  TYPE ORACLE
  VERSION 8.0.3
  OPTIONS
     (ADD FOLD_ID 'N',
     SET CPU_RATIO '2.0')
```

- *Example 2:* Indicate that the Documentum data source called DCTM_SVR_ASIA has been changed to Version 4.

```
ALTER SERVER DCTM_SVR_ASIA
  VERSION 4
```

ALTER SERVICE CLASS

The ALTER SERVICE CLASS statement alters the definition of a service class.

Invocation

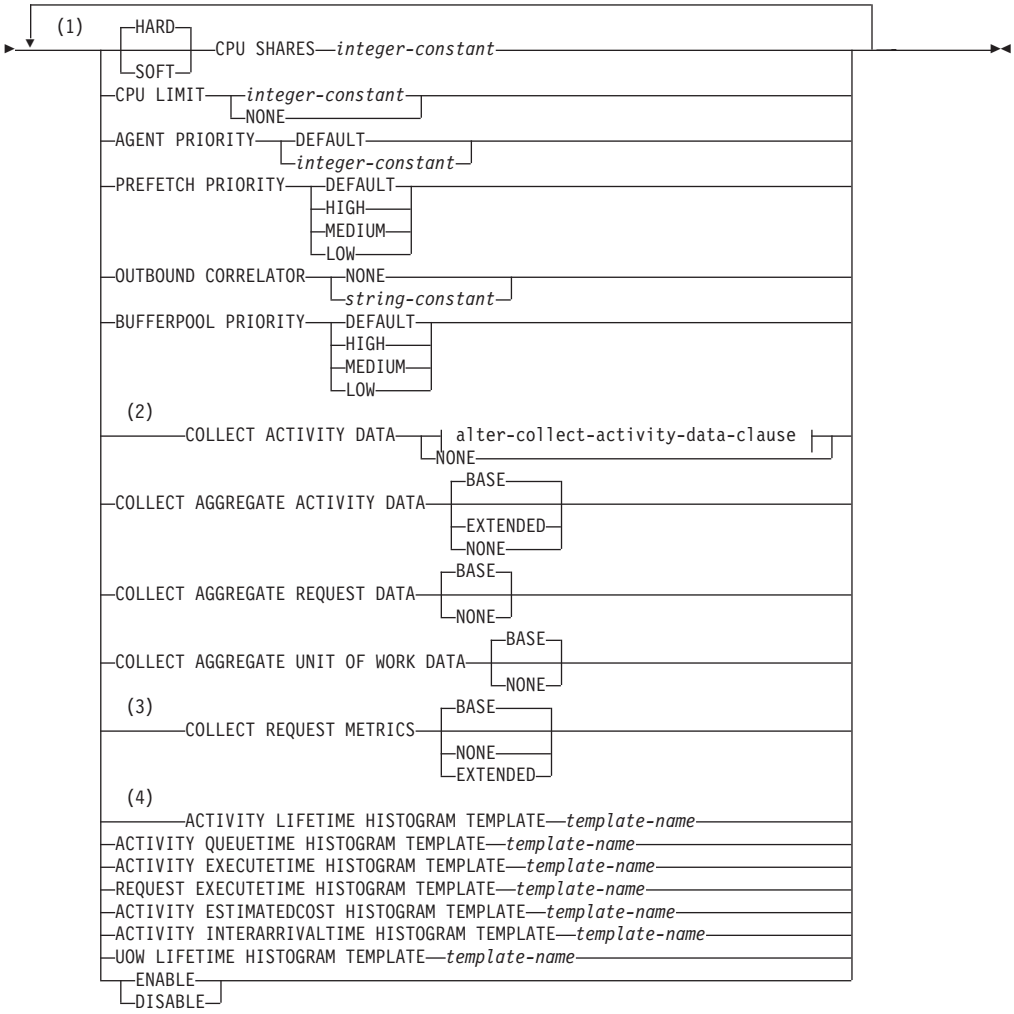
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

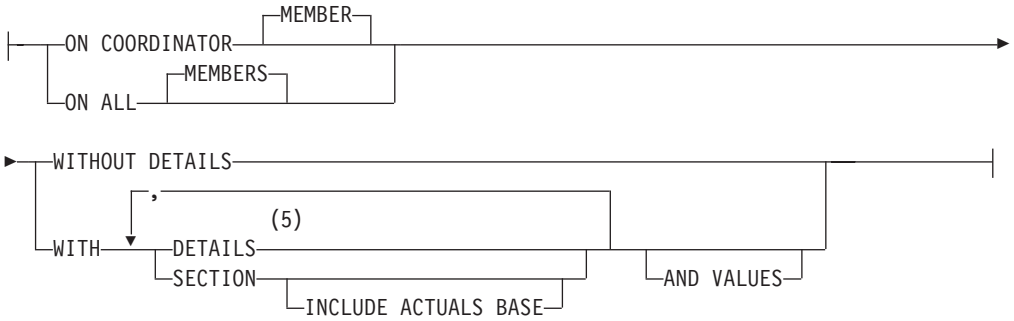
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- SQLADM authority, only if every alteration clause is a COLLECT clause
- WLMADM authority
- DBADM authority

Syntax



alter-collect-activity-data-clause:



Notes:

1 The same clause must not be specified more than once.

2 All COLLECT clauses except for COLLECT REQUEST METRICS are only valid for a service subclass.

3 The COLLECT REQUEST METRICS clause is only valid for a service superclass.

4 The HISTOGRAM TEMPLATE clauses are only valid for a service subclass.

5 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

Description

service-class-name
Identifies the service class that is to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited).The *service-class-name* must identify a service class that exists in the database (SQLSTATE 42704). To alter a service subclass, the *service-superclass-name* must be specified using the UNDER clause.

UNDER *service-superclass-name*
This clause is used only for altering a service subclass. The *service-superclass-name* identifies the service superclass of the service subclass and must identify a service superclass that exists in the database (SQLSTATE 42704).

SOFT CPU SHARES *integer-constant* **or HARD CPU SHARES** *integer-constant*
Specifies the number of shares of CPU resources that the WLM dispatcher allocates to this service class when work is executing within this service class. Valid values for the *integer-constant* are integers between 1 and 65535. Qualifying **CPU SHARES** with the keyword **HARD**, or specifying **CPU SHARES** without qualifying it with the keyword **HARD** or **SOFT**, indicates that hard CPU shares are to be allocated to this service class. Specifying the keyword **SOFT** indicates that soft CPU shares are to be allocated to this service class. To use hard and soft CPU shares with DB2 workload manager dispatcher, you must enable the **wlm_disp_cpu_shares** database manager configuration parameter.

CPU LIMIT *integer-constant* **or CPU LIMIT NONE**
Specifies the maximum percentage of the CPU resources that the WLM dispatcher can assign to this service class. Valid values for the *integer-constant* are integers between 1 and 100. You can also specify CPU LIMIT NONE to indicate that there is no CPU limit.

AGENT PRIORITY DEFAULT or AGENT PRIORITY *integer-constant*
Specifies the relative (delta) operating system priority of agents running in the service class or the normal priority of threads running in DB2. The default value is DEFAULT.

Important: The **agentpri** database manager configuration is deprecated since Version 9.5. It can still be used in pre-Version 9.5 data servers and clients. Also, agent priority for the WLM service class has been deprecated in Version 10.1 and might be removed in a future release. Start to use the WLM dispatcher capability instead of agent priority. For more information, see "Agent priority of service classes has been deprecated" in *What's New for DB2 Version 10.1*.

When set to DEFAULT, no special action is taken, and agents in the service class are scheduled according to the normal priority that the operating system schedules all DB2 threads. When this parameter is set to a value other than DEFAULT, agents are set to a priority that is equal to the normal priority plus AGENT PRIORITY when the next activity begins. For example, if the normal priority is 20 and AGENT PRIORITY is set to -10, the priority of agents in the service class is set to 20 - 10 = 10.

Note: Agent priority and WLM dispatcher shares cannot be used together. When the dispatcher is enabled by setting the value of the **wlm_dispatcher**

database manager configuration parameter to ON, the specified agent priority setting is ignored and agent priority is set to the default value until the dispatcher is disabled.

DB2 workload manager (WLM) does not assign service class agent priority to work being done within a fenced mode process (FMP). Fenced procedures do not run their logic within a service class. These fenced procedures run within the DB2 FMP and this work is not done by DB2 agents. As a reminder, DB2 WLM controls DB2 agents.

On UNIX operating systems and Linux, valid values are DEFAULT and -20 to 20 (SQLSTATE 42615). Negative values denote a higher relative priority. Positive values denote a lower relative priority.

On Windows operating systems, valid values are DEFAULT and -6 to 6 (SQLSTATE 42615). Negative values denote a lower relative priority. Positive values denote a higher relative priority.

If AGENT PRIORITY is DEFAULT for a service subclass, it inherits the AGENT PRIORITY value of its parent superclass. AGENT PRIORITY cannot be altered for a default subclass (SQLSTATE 5U032). AGENT PRIORITY must be set to DEFAULT if OUTBOUND CORRELATOR is set (SQLSTATE 42613).

Note: On AIX®, the instance owner must have CAP_NUMA_ATTACH and CAP_PROPAGATE capabilities to set a higher relative priority for agents in a service class using AGENT PRIORITY. To grant these capabilities, logon as root and run the following command:

```
chuser capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE
```

On Solaris 10 or higher, the instance owner must have the proc_priocntl privilege to set a higher relative priority for agents in a service class using AGENT PRIORITY. To grant this privilege, logon as `root` and run the following command:

```
usermod -K defaultpriv=basic,proc_priocntl db2user
```

In this example, proc_priocntl is added to the default privilege set of user db2user.

Moreover, when DB2 is running in a non-global zone of Solaris, the proc_priocntl privilege must be added to the zone's limit privilege set. To grant this privilege to the zone, logon as `root` and run the following command:

```
global# zonecfg -z db2zone
zonecfg:db2zone> set limitpriv="default,proc_priocntl"
```

In this example, proc_priocntl is added to the limit privilege set of zone db2zone.

On Solaris 9, there is no facility for DB2 to raise the relative priority of agents. Upgrade to Solaris 10 or higher to use the service class agent priority.

PREFETCH PRIORITY DEFAULT | HIGH | MEDIUM | LOW

This parameter controls the priority with which agents in the service class can submit their prefetch requests. Valid values are HIGH, MEDIUM, LOW, or DEFAULT (SQLSTATE 42615). HIGH, MEDIUM, and LOW mean that prefetch requests will be submitted to the high, medium, and low priority queues, respectively. Prefetchers empty the priority queue in order from high to low. Agents in the service class submit their prefetch requests at the PREFETCH PRIORITY level when the next activity begins. If PREFETCH PRIORITY is altered after a prefetch request is submitted, the request priority does not change. The default value is DEFAULT, which is internally mapped to

MEDIUM for service superclasses. If DEFAULT is specified for a service subclass, it inherits the PREFETCH PRIORITY of its parent superclass.

PREFETCH PRIORITY cannot be altered for a default subclass (SQLSTATE 5U032).

OUTBOUND CORRELATOR NONE or OUTBOUND CORRELATOR *string-constant*
Specifies whether or not to associate threads from this service class to an external workload manager service class.

If OUTBOUND CORRELATOR is set to a *string-constant* for the service superclass and OUTBOUND CORRELATOR NONE is set for a service subclass, the service subclass inherits the OUTBOUND CORRELATOR of its parent. OUTBOUND CORRELATOR must be set to NONE if the AGENT PRIORITY is not set to DEFAULT (SQLSTATE 42613).

OUTBOUND CORRELATOR NONE
For a service superclass, specifies that there is no external workload manager service class association with this service class, and for a service subclass, specifies that the external workload manager service class association is the same as its parent.

OUTBOUND CORRELATOR *string-constant*
Specifies the *string-constant* that is to be used as a correlator to associate threads from this service class to an external workload manager service class. The external workload manager must be active (SQLSTATE 5U030). The external workload manager should be set up to recognize the value of *string-constant*.

BUFFERPOOL PRIORITY DEFAULT | HIGH | MEDIUM | LOW
This parameter controls the bufferpool priority of pages fetched by activities in this service class. Valid values are HIGH, MEDIUM, LOW or DEFAULT (SQLSTATE 42615). Pages fetched by activities in a service class with higher bufferpool priority are less likely to be swapped out than pages fetched by activities in a service class with lower bufferpool priority. If DEFAULT is specified for a service subclass, it inherits the BUFFERPOOL PRIORITY from its parent superclass.

BUFFERPOOL PRIORITY cannot be altered for a default subclass (SQLSTATE 5U032).

COLLECT ACTIVITY DATA
Specifies that information about each activity that executes in this service class is to be sent to any active activities event monitor when the activity completes. The COLLECT ACTIVITY DATA clause is only valid for a service subclass.

alter-collect-activity-data-clause

ON COORDINATOR MEMBER
Specifies that activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that activity data is to be collected at all members where the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. If the AND VALUES clause is specified, activity input values will be collected only for the members of the coordinator.

WITHOUT DETAILS
Specifies that data about each activity that executes in the service class is to be sent to any active activities event monitor, when the activity

completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. Section actuals will be collected on any partition where the activity data is collected.

INCLUDE ACTUALS BASE
Specifies that section actuals should also be collected on any partition where the activity data is collected. For section actuals to be collected, either INCLUDE ACTUALS clause must be specified or the **section_actuals** database configuration parameter must be set.

The effective setting for the collection of section actuals is the combination of the INCLUDE ACTUALS clause (specified on the WORK ACTION, SERVICE CLASS, or WORKLOAD), the **section_actuals** database configuration parameter, and the <collectsectionactuals> setting specified on the WLM_SET_CONN_ENV routine. For example, if INCLUDE ACTUALS BASE is specified, yet the **section_actuals** database configuration parameter value is NONE and <collectsectionactuals> is set to NONE, then the effective setting for the collection of section actuals is BASE.

BASE specifies that the following actuals should be enabled and collected during the activity's execution:

- Basic operator cardinality counts
- Statistics for each object referenced (DML statements only)

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

NONE
Specifies that activity data should not be collected for each activity that executes in this service class.

COLLECT AGGREGATE ACTIVITY DATA
Specifies that aggregate activity data should be captured for this service class and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default is COLLECT AGGREGATE ACTIVITY DATA BASE. The COLLECT AGGREGATE ACTIVITY DATA clause is only valid for a service subclass.

BASE
Specifies that basic aggregate activity data should be captured for this service class and sent to the statistics event monitor, if one is active. Basic aggregate activity data includes:

- Estimated activity cost high watermark
- Rows returned high watermark
- Temporary table space usage high watermark

  Note: Only activities that have an SQLTEMPSPACE threshold applied to them participate in this high watermark.
- Activity life time histogram
- Activity queue time histogram
- Activity execution time histogram

EXTENDED
Specifies that all aggregate activity data should be captured for this service class and sent to the statistics event monitor, if one is active. This includes all basic aggregate activity data plus:

- Activity data manipulation language (DML) estimated cost histogram
- Activity DML inter-arrival time histogram

NONE
Specifies that no aggregate activity data should be captured for this service class.

COLLECT AGGREGATE REQUEST DATA
Specifies that aggregate request data should be captured for this service class and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval specified by the **wlm_collect_int** database configuration parameter. The default is COLLECT AGGREGATE REQUEST DATA NONE. The COLLECT AGGREGATE REQUEST DATA clause is valid only for a service subclass.

BASE
Specifies that basic aggregate request data should be captured for this service class and sent to the statistics event monitor, if one is active.

NONE
Specifies that no aggregate request data should be captured for this service class.

COLLECT AGGREGATE UNIT OF WORK DATA
Specifies that aggregate unit of work data is to be captured for this service class and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default, when COLLECT AGGREGATE UNIT OF WORK DATA is specified, is COLLECT AGGREGATE UNIT OF WORK DATA BASE.

BASE
Specifies that basic aggregate unit of work data is to be captured for this service class and sent to the statistics event monitor, if one is active. Basic aggregate unit of work data includes:

- Unit of work lifetime histogram

NONE
Specifies that no aggregate unit of work data is to be collected for this service class.

COLLECT REQUEST METRICS
Specifies that monitor metrics should be collected for any request submitted by a connection that is associated with the specified service superclass and sent to the statistics and unit of work event monitors, if active. The default is COLLECT REQUEST METRICS NONE. The COLLECT REQUEST METRICS clause is only valid for a service superclass (SQLSTATE 50U44).

Note: The effective request metrics collection setting is the combination of the attribute specified by the COLLECT REQUEST METRICS clause on the service superclass associated with the connection submitting the request, and the **mon_req_metrics** database configuration parameter. If either the service superclass attribute or the configuration parameter has a value other than NONE, metrics will be collected for the request.

BASE
Specifies that basic metrics will be collected for any request submitted by a connection associated with the service superclass.

EXTENDED
Specifies that basic metrics will be collected for any request submitted by a connection associated with the service superclass. In addition, specifies that the values for the following monitor elements should be determined with additional granularity:

- **total_section_time**
- **total_section_proc_time**
- **total_routine_user_code_time**
- **total_routine_user_code_proc_time**
- **total_routine_time**

NONE
Specifies that no metrics will be collected for any request submitted by a connection associated with the service superclass.

ACTIVITY LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of DB2 activities running in the service class during a specific interval. This time includes both time queued and time executing. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option. This clause is only valid for a service subclass.

ACTIVITY QUEUETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the service class are queued during a specific interval. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option. This clause is only valid for a service subclass.

ACTIVITY EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the service class are executing during a specific interval. This time does not include the time spent queued. Activity execution time is collected in this histogram at the coordinator member only. The time does not include idle time. Idle time is the time between the execution of requests belonging to the same activity when no work is being done. An example of idle time is the time between the end of opening a cursor and the start of fetching from that cursor.

This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option. This clause is only valid for a service subclass.

REQUEST EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 requests running in the service class are executing during a specific interval. This time does not include the time spent queued. Request execution time is collected in this histogram on each member where the request executes. This information is only collected when the COLLECT AGGREGATE REQUEST DATA clause is specified with the BASE option. This clause is only valid for a service subclass.

ACTIVITY ESTIMATEDCOST HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the estimated cost, in timerons, of DML activities running in the service class. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option. This clause is only valid for a service subclass.

ACTIVITY INTERARRIVALTIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, between the arrival of one DML activity and the arrival of the next DML activity. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option. This clause is only valid for a service subclass.

UOW LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of units of work running in the service class during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE UNIT OF WORK DATA clause is specified with the BASE option.

ENABLE or DISABLE
Specifies whether or not connections and activities can be mapped to the service class.

ENABLE
Connections and activities can be mapped to the service class.

DISABLE
Connections and activities cannot be mapped to the service class. New connections or activities that are mapped to a disabled service class will be rejected (SQLSTATE 5U028). When a service superclass is disabled, its service subclasses are also disabled. When the service superclass is re-enabled, its service subclasses return to states that are defined in the system catalog. A default service class cannot be disabled (SQLSTATE 5U032).

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (histogram template)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (service class)

  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (threshold)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (work action set)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (work class set)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (workload)
  - GRANT (workload privileges) or REVOKE (workload privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all members. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until after a COMMIT statement, even for the connection that issues the statement.
- After the ALTER SERVICE CLASS statement is committed, changes to AGENT PRIORITY, PREFETCH PRIORITY, OUTBOUND CORRELATOR, and COLLECT take effect for the next new activity in the service class. Existing activities in the service class continue to complete their work using the old settings.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - o DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - o DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

Examples

- *Example 1:* Alter the agent priority of agents in service superclass PETSALES to raise it from DEFAULT to the highest possible value (shown for UNIX and Linux operating systems; on Windows operating systems, substitute 6).

```
ALTER SERVICE CLASS PETSALES AGENT PRIORITY -20
```

- *Example 2:* Alter service superclass BARNSALES and add an outbound correlator 'osLowPriority'. Threads running in the service superclass and its service subclasses will have the outbound correlator 'osLowPriority' associated with them.

```
ALTER SERVICE CLASS BARNSALES OUTBOUND CORRELATOR 'osLowPriority'
```

ALTER STOGROUP

The ALTER STOGROUP statement is used to alter the definition of a storage group.

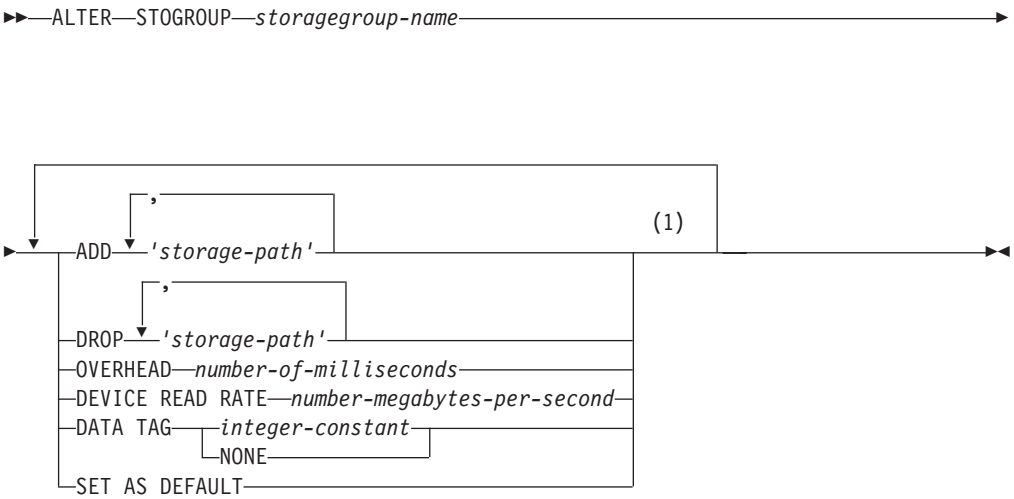
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax



Notes:

1 Each clause can be specified only once.

Description

storagegroup-name
Identifies the storage group to be altered; *storagegroup-name* must identify a storage group that exists at the current server (SQLSTATE 42704). This is a one-part name.

ADD
Specifies that one or more new storage paths are to be added to the specified storage group.

storage-path
A string constant that specifies either an absolute path or the letter name of a drive (Windows operating systems only) on which containers for automatic storage table spaces are to be created. The string can include database partition expressions to specify database partition number information in the storage path. For predictable performance, ensure the storage paths added to a storage group have similar media characteristics.

The maximum length of a storage path is 175 characters (SQLSTATE 54036). A storage path being added must be valid according to the naming rules for paths, and must be accessible (SQLSTATE 57019). Similarly, in a partitioned database environment, the storage path must exist and be accessible on every database partition (SQLSTATE 57019).

DROP
Specifies that one or more storage paths are to be removed from the given storage group. If table spaces are actively using a storage path being dropped, then the state of the storage path is changed from "In Use" to "Drop Pending" and future use of the storage path will be prevented.

The DROP *storage-path* clause is not supported in a DB2 pureScale environment (SQLSTATE 56038).

storage-path
A string constant that specifies either an absolute path or the letter name of a drive (Windows operating systems only). The string can include database partition expressions to specify database partition number information in the storage path.

A storage path being dropped must currently exist in the storage group (SQLSTATE 57019) and cannot already be in the "Drop Pending" state (SQLSTATE 55073).

OVERHEAD *number-of-milliseconds*
Specifies the I/O controller usage and disk seek and latency time. This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all storage paths, set the value to a numeric literal which represents the average for all storage paths that belong to the storage group.

DEVICE READ RATE *number-megabytes-per-second*
Represents the device specification for the read transfer rate in megabytes per second. This value is used to determine the cost of I/O during query optimization. The value of *number-megabytes-per-second* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all storage paths, set the value to a numeric literal which represents the average for all storage paths that belong to the storage group.

DATA TAG *integer-constant* **or DATA TAG NONE**
Specifies a tag for the data in a given storage group. This value can be used as part of a WLM configuration in a work class definition or referenced within a threshold definition. For more information, see the CREATE WORK CLASS SET, ALTER WORK CLASS SET, CREATE THRESHOLD, and ALTER THRESHOLD statements.

integer-constant
Valid values for *integer-constant* are integers from 1 to 9.

NONE
If NONE is specified, there is no data tag.

SET AS DEFAULT
Specifies that the storage group being altered is designated as the default storage group. There can be only one storage group designated as the default storage group. There is no affect to the existing table spaces using that storage group. The designated default storage group is used by automatic storage table

spaces when no storage group is specified at table space creation and a database managed table space is converted to automatic storage managed during redirected restore.

Rules

- A storage group must have at least one storage path. Dropping all storage paths from the storage group is not permitted (SQLSTATE 428HH).
- The ALTER STOGROUP statement cannot be executed while a database partition server is being added (SQLSTATE 55071).
- A storage group can have up to 128 defined storage paths (SQLSTATE 5U009).
- A transaction can have at most one ALTER STOGROUP statement per storage group. In the case of the default storage group, there can be at most one ALTER DATABASE statement or one ALTER STOGROUP statement on the default storage group (SQLSTATE 25502).

Notes

- ***Adding new storage paths:*** When adding new storage paths:
  - Existing REGULAR and LARGE table spaces using this storage group will not initially use these new paths. The database manager might choose to create new table space containers on these paths only if an out-of-space condition occurs. You can issue ALTER TABLESPACE REBALANCE statements for existing table spaces to stripe them over the newly added storage paths.
  - Existing temporary table spaces managed by automatic storage do not automatically use new storage paths. The database must be stopped normally then restarted for containers in these table spaces to use the new storage path or paths. As an alternative, the temporary table spaces can be dropped and re-created. When created, these table spaces automatically use all storage paths that have sufficient free space.
- ***Calculation of free space:*** When free space is calculated for a storage path on a database partition, the database manager checks for the existence of the following directories or mount points within the storage path, and will use the first one that is found.

  ```
  <storage path>/<instance name>/NODE####/<database name>
  <storage path>/<instance name>/NODE####
  <storage path>/<instance name>
  <storage path>
  ```

  Where:
  - `<storage path>` is a storage path associated with the database.
  - `<instance name>` is the instance under which the database resides.
  - `NODE####` corresponds to the database partition number (for example, NODE0000 or NODE0001).
  - `<database name>` is the name of the database.
- ***Isolating multiple database partitions under one storage path:*** File systems can be mounted at a point beneath the storage path, and the database manager will recognize that the actual amount of free space available for table space containers might not be the same amount that is associated with the storage path directory itself.

  Consider an example in which two logical database partitions exist on one physical computer, and there is a single storage path (`/db2data`). Each database partition will use this storage path, but you might want to isolate the data from each partition within its own file system. In this case, a separate file system can be created for each partition and it can be mounted at `/db2data/<instance>/`

NODE####. When creating containers on the storage path and determining free space, the database manager will not retrieve free space information for /db2data, but instead will retrieve it for the corresponding /db2data/<instance>/NODE#### directory.

- ***Dropping a storage path that is in use by one or more table spaces:*** When dropping a storage path that is in use by one or more table spaces, the state of the path changes from "In Use" to "Drop Pending". Future growth on the path will not occur. Before the path can be fully removed from the storage group, each affected table space must be rebalanced (using the REBALANCE clause of the ALTER TABLESPACE statement) so that its container data is moved off the storage path. Rebalance is supported only for REGULAR and LARGE table spaces. Drop and re-create temporary table spaces to have their containers removed from the dropped path. When the path is no longer in use by any table space, it will be physically removed from the database.

  For a partitioned database environment, the path is maintained independently on each partition. When a path is no longer in use on a given database partition, it will be physically removed from that partition. Other partitions might still show the path as being in the "Drop Pending" state. The list of automatic storage table spaces using drop pending storage paths can be determined by issuing the following SQL statement:

  ```
  SELECT DISTINCT TBSP_NAME, TBSP_ID, TBSP_CONTENT_TYPE
    FROM TABLE(MON_GET_TABLESPACE(NULL,-2)) AS T
    WHERE TBSP_PATHS_DROPPED = 1
  ```

- ***Dropping a storage path that was added to a storage group multiple times:*** It is possible for a given storage path to be added to a storage group multiple times. When using the DROP clause, specifying that particular path once will drop all instances of the path from the storage group.

Examples

- *Example 1*: Add drives D and E to the storage group named COMPLIANCE.

  ```
  ALTER STOGROUP COMPLIANCE ADD 'D:\', 'E:\'
  ```

- *Example 2*: Change the data tag for the OPERATIONAL storage group and designate it as the default storage group.

  ```
  ALTER STOGROUP OPERATIONAL DATA TAG 3 SET AS DEFAULT
  ```

- *Example 3*: Add a storage path that uses a database partition expression to differentiate the storage paths on each of the database partitions.

  ```
  ALTER STOGROUP TESTDATA ADD '/dataForPartition $N'
  ```

- *Example 4*: Remove paths /db2/filesystem1 and /db2/filesystem2 from storage group TESTDATA.

  ```
  ALTER STOGROUP TESTDATA DROP '/db2/filesystem1', '/db2/filesystem2'
  ```

ALTER TABLE

The ALTER TABLE statement alters the definition of a table.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTER privilege on the table to be altered
- CONTROL privilege on the table to be altered
- ALTERIN privilege on the schema of the table
- DBADM authority

To create or drop a foreign key, the privileges held by the authorization ID of the statement must include one of the following authorities on the parent table:

- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- DBADM authority

To drop the primary key or a unique constraint on table T, the privileges held by the authorization ID of the statement must include at least one of the following authorities on every table that is a dependent of this parent key of T:

- ALTER privilege on the table
- CONTROL privilege on the table
- ALTERIN privilege on the schema of the table
- DBADM authority

To alter a table to become a materialized query table (using a fullselect), the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the table
- DBADM authority

and at least one of the following authorities on each table or view identified in the fullselect (excluding group privileges):

- SELECT privilege and ALTER privilege (including group privileges) on the table or view
- CONTROL privilege on the table or view
- SELECT privilege on the table or view, and ALTERIN privilege (including group privileges) on the schema of the table or view
- DATAACCESS authority

To alter a table so that it is no longer a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following authorities on each table or view identified in the fullselect used to define the materialized query table:

- ALTER privilege on the table or view
- CONTROL privilege on the table or view
- ALTERIN privilege on the schema of the table or view
- DBADM authority

To add a column of type DB2SECURITYLABEL to a table, the privileges held by the authorization ID of the statement must include at least a security label from the security policy associated with the table.

To remove the security policy from a table, the privileges held by the authorization ID of the statement must include SECADM authority.

To alter a table to attach a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following authorities on the source table:

- SELECT privilege on the table and DROPIN privilege on the schema of the table
- CONTROL privilege on the table
- DATAACCESS authority

and at least one of the following authorities on the target table:

- ALTER and INSERT privileges on the table
- CONTROL privilege on the table
- DATAACCESS authority

To alter a table to detach a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following authorities on the target table of the detached partition:

- CREATETAB authority on the database, and USE privilege on the table spaces used by the table, as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the new table does not exist
  - CREATEIN privilege on the schema, if the schema name of the new table refers to an existing schema
- DBADM authority

and at least one of the following authorities on the source table:

- SELECT, ALTER, and DELETE privileges on the table
- CONTROL privilege on the table
- DATAACCESS authority

To alter a table to activate not logged initially with empty table, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTER and DELETE privileges on the table
- CONTROL privilege on the table
- DBADM authority

To alter a table that is protected by a security policy to activate not logged initially with empty table, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the table
- DBADM authority

To alter a table to ACTIVATE and DEACTIVATE row and column access control, the privileges held by the authorization ID of the statement must include the SECADM authority.

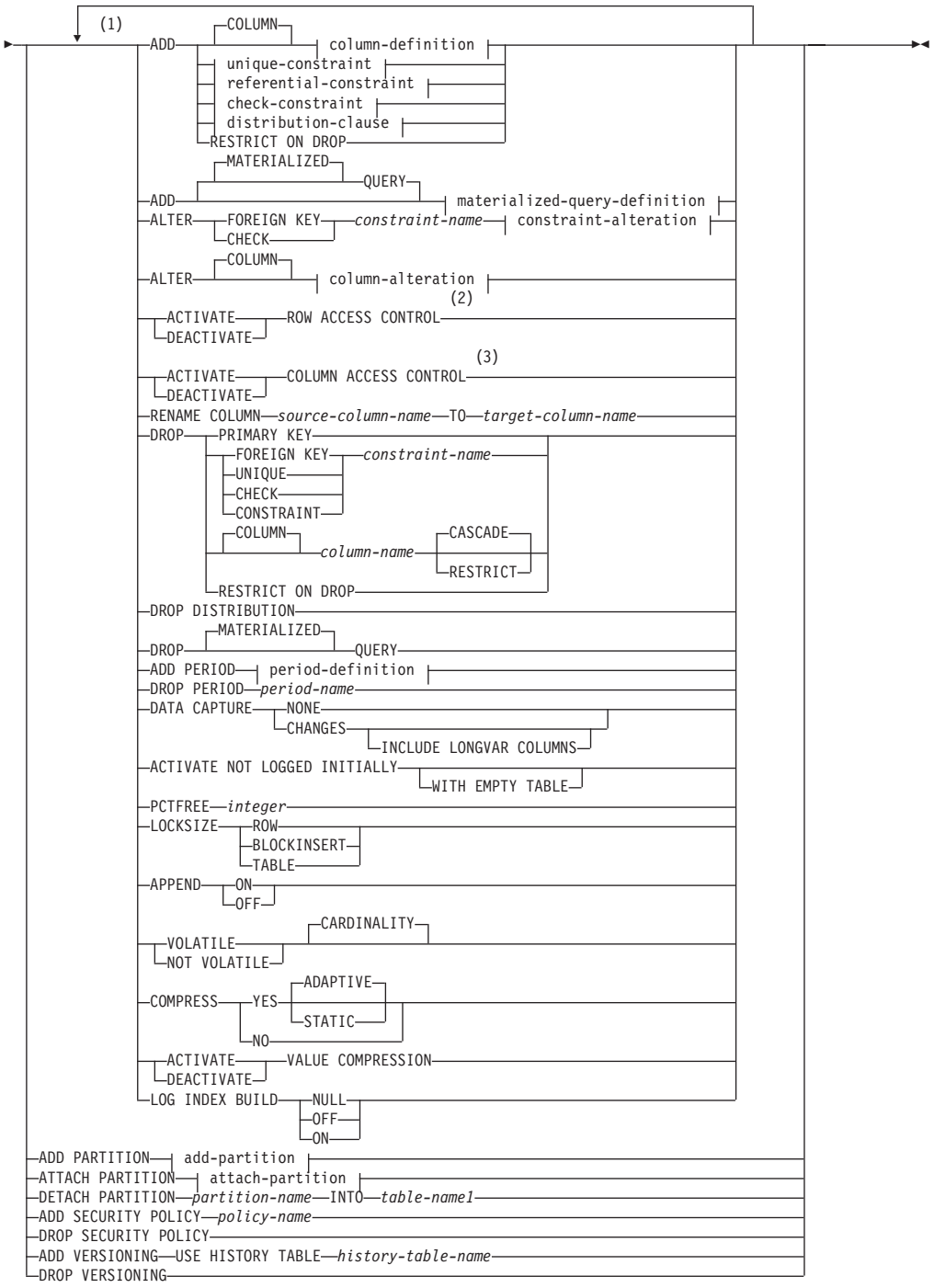
To alter a table with ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE, if that table has row access control activated, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the table
- DBADM authority

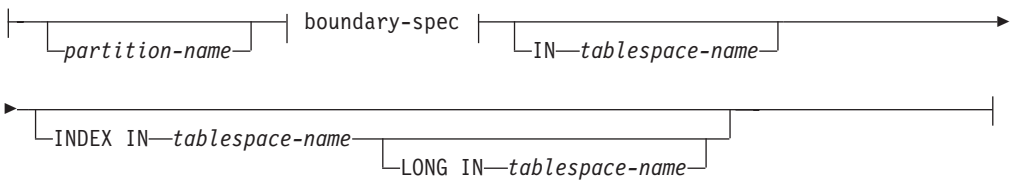
To alter a table to become a system-period temporal table (with the ADD VERSIONING clause) or alter a system-period temporal table when one or more of the changes also result in changes to the associated history table, the privileges that are held by the authorization ID of the statement must also include at least one of the following authorities:

- ALTER privilege on the history table
- CONTROL privilege on the history table
- ALTERIN privilege on the schema of the history table
- DBADM authority

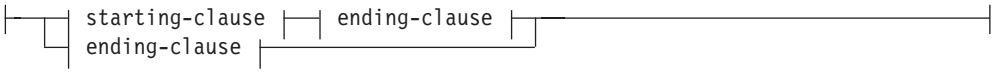
Syntax



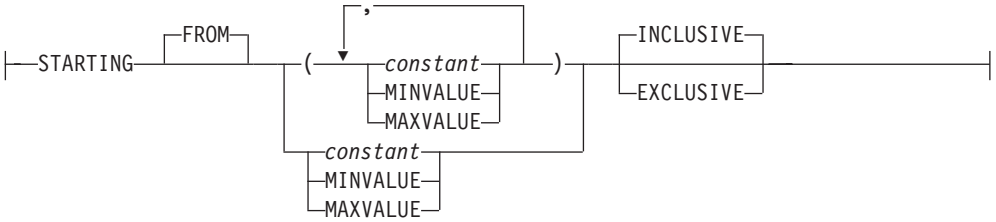
add-partition:



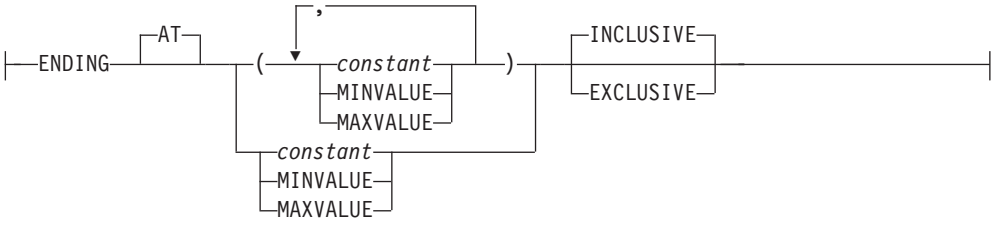
boundary-spec:



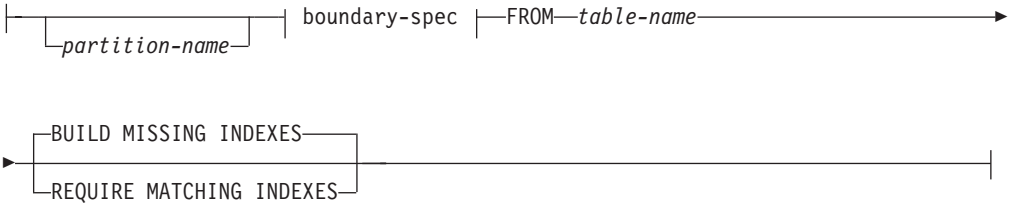
starting-clause:



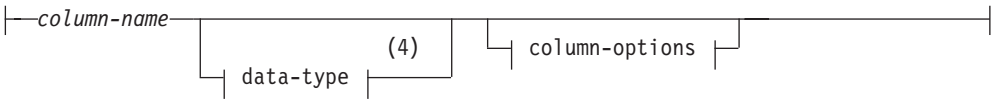
ending-clause:



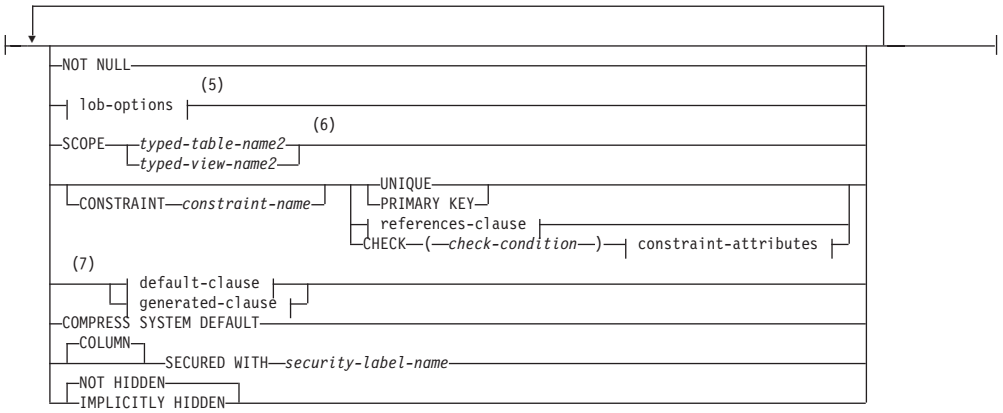
attach-partition:



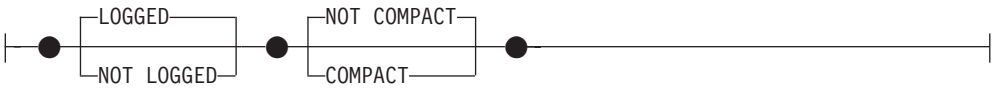
column-definition:



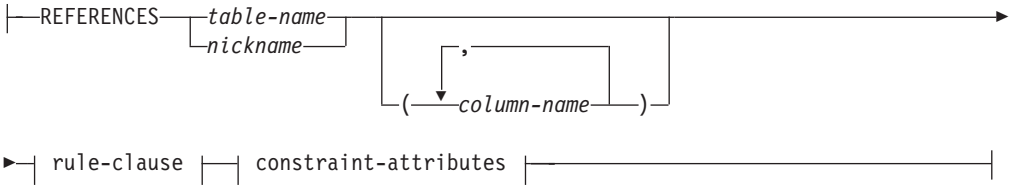
column-options:



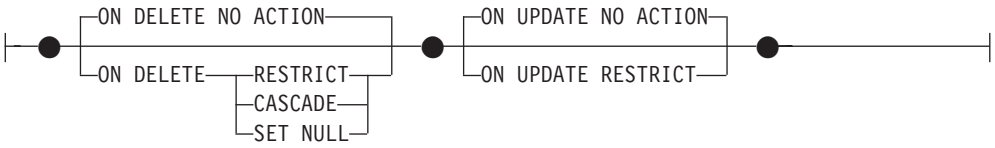
lob-options:



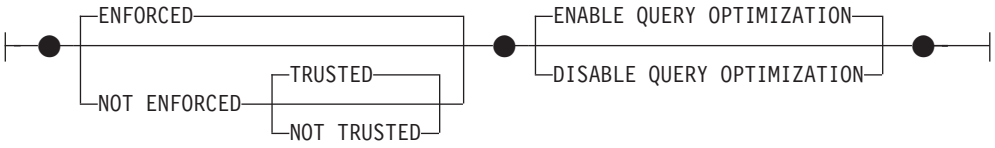
references-clause:



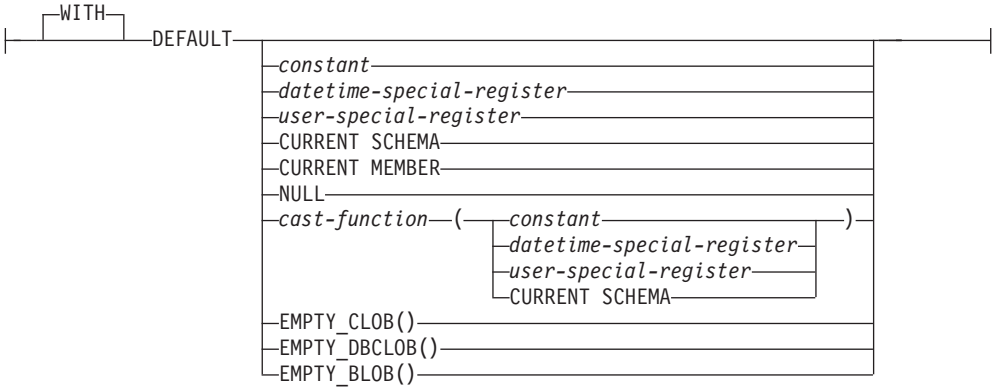
rule-clause:



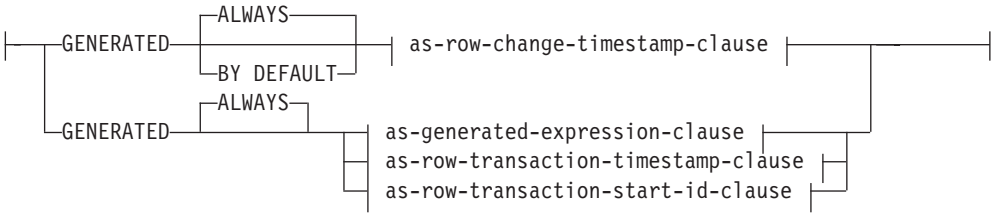
constraint-attributes:



default-clause:



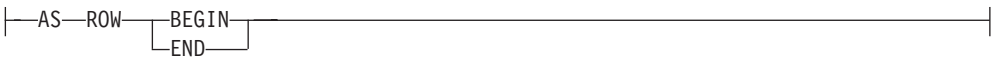
generated-clause:



as-row-change-timestamp-clause:

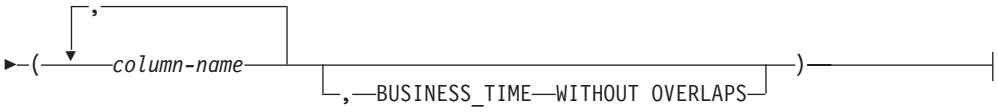
as-generated-expression-clause:

as-row-transaction-timestamp-clause:



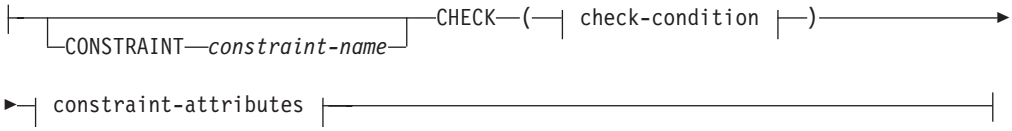
as-row-transaction-start-id-clause:

unique-constraint:



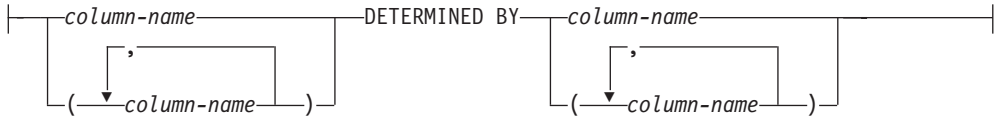
referential-constraint:

check-constraint:

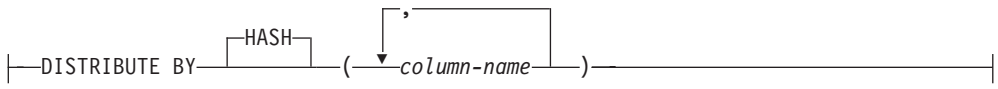


check-condition:

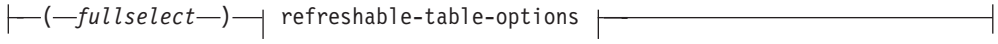
functional-dependency:



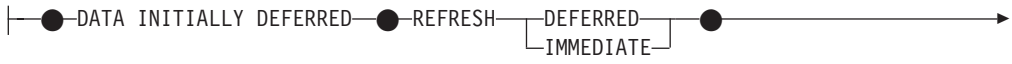
distribution-clause:



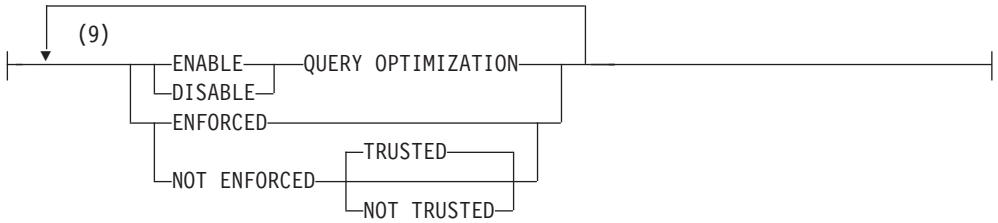
materialized-query-definition:



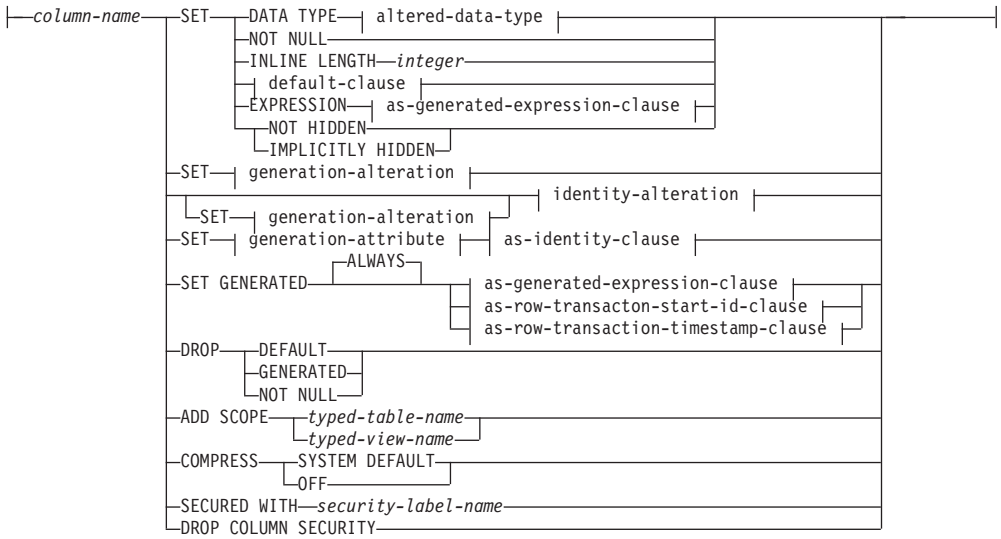
refreshable-table-options:



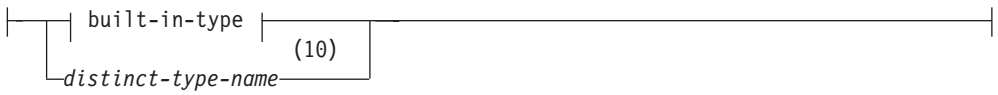
constraint-alteration:



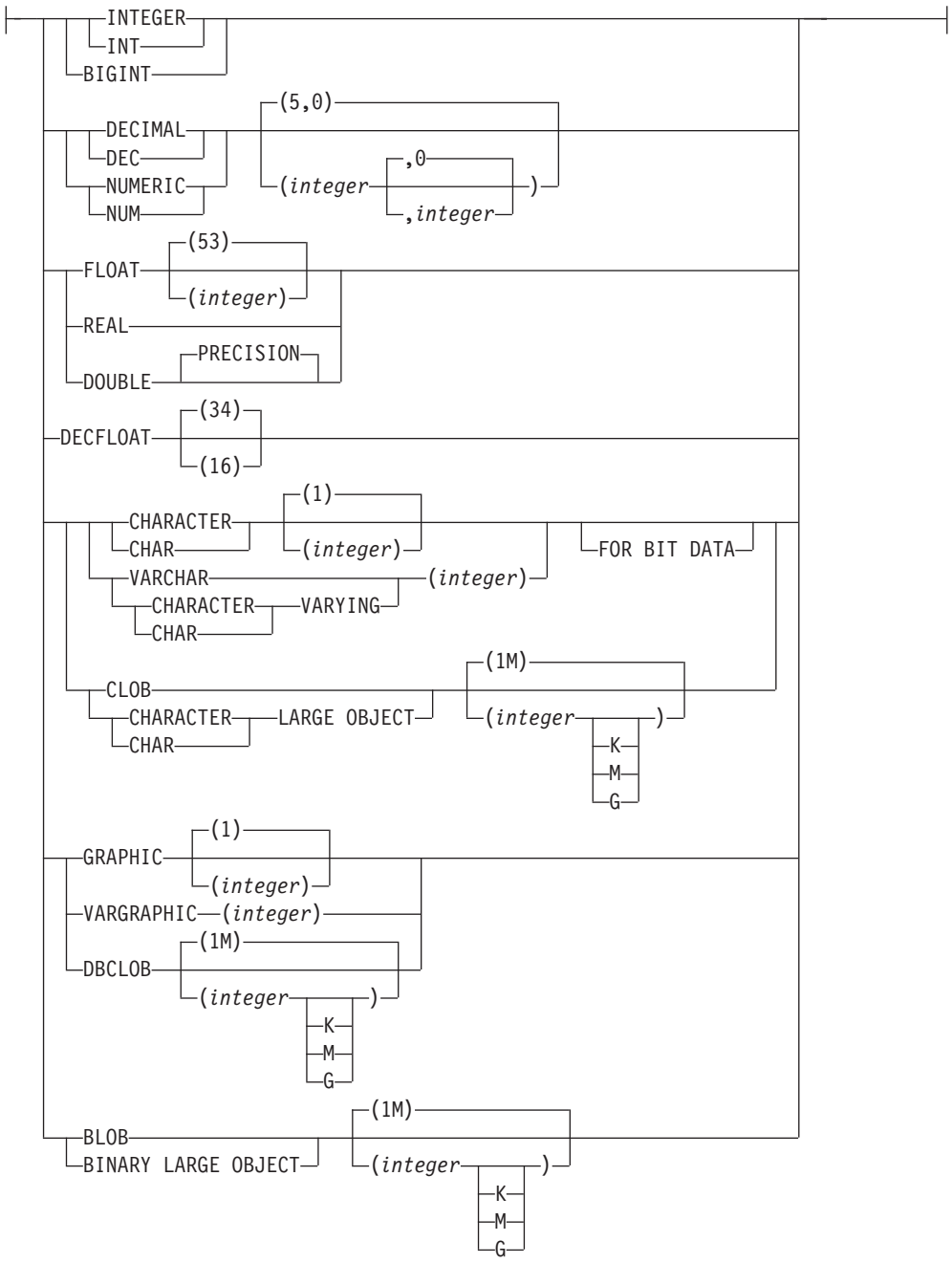
column-alteration:



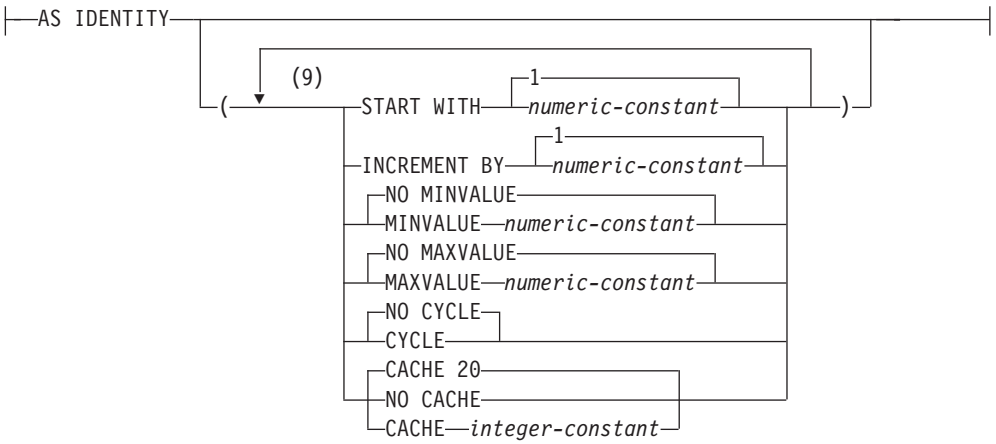
altered-data-type:



built-in-type:

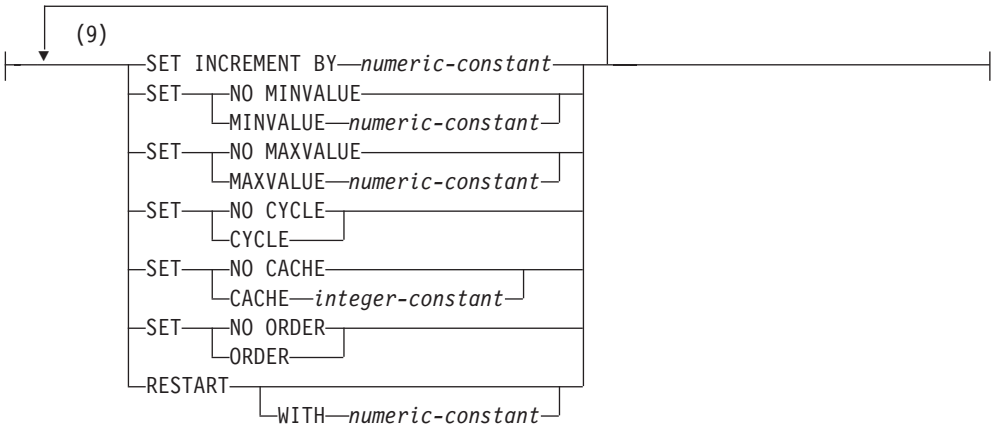


as-identity-clause:

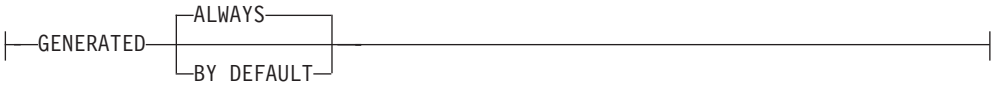


generation-alteration:

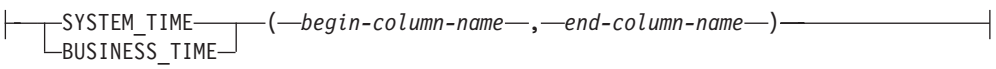
identity-alteration:



generation-attribute:



period-definition:



Notes:

1 The same clause must not be specified more than once (SQLSTATE 42614).

2 If an ACTIVATE or DEACTIVATE clause is specified for row access control,

no other clause except ACTIVATE or DEACTIVATE column access control can be specified in the same ALTER TABLE statement (SQLSTATE 42613).

3 If an ACTIVATE or DEACTIVATE clause is specified for column access control, no other clause except ACTIVATE or DEACTIVATE row access control can be specified in the same ALTER TABLE statement (SQLSTATE 42613).

4 If the first column option chosen is *generated-clause*, *data-type* can be omitted; it will be computed by the generation expression.

5 The *lob-options* clause only applies to large object types (CLOB, DBCLOB, and BLOB), and to distinct types that are based on large object types.

6 The SCOPE clause only applies to the REF type.

7 The default-clause and generated-clause cannot both be specified for the same column definition (SQLSTATE 42614).

8 Data type is optional for a row change timestamp column if the first column-option specified is a generated-clause; the data type default is TIMESTAMP(6). Data type is optional for row-begin, row-end, and transaction-start-ID columns if the first column-option is a generated-clause; the data type default is TIMESTAMP(12)

9 The same clause must not be specified more than once.

10 The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type (SQLSTATE 428H2).

Description

table-name
The *table-name* must identify a table that exists at the current server. It cannot be a nickname (SQLSTATE 42809) and must not be a view, a catalog table, a created temporary table, or a declared temporary table (SQLSTATE 42995).

If *table-name* identifies a materialized query table, alterations are limited to adding or dropping the materialized query, invoking the ACTIVATING NOT LOGGED INITIALLY clause, adding or dropping RESTRICT ON DROP, modifying data capture, pctfree, locksize, append, volatile, data row compression, value compression, and activating or deactivating row and column access control.

If *table-name* identifies a range-clustered table, alterations are limited to adding, changing, or dropping constraints, activating not logged initially, adding or dropping RESTRICT ON DROP, changing locksize, data capture, or volatile, and setting column default values.

ADD *column-definition*
Adds a column to the table. The table must not be a history table for a system-period temporal table (SQLSTATE 428HZ) or a typed table (SQLSTATE 428DH). For all existing rows in the table, the value of the new column is set to its default value. The new column is the last column of the table; that is, if initially there are *n* columns, the added column is column *n*+1.

Adding the new column must not make the total byte count of all columns exceed the maximum record size.

If the table is a system-period temporal table, the column is added to the associated history table as well.

If the added column is a generated column that is based on an expression, the expression must not reference a column for which a column mask is defined (SQLSTATE 42621).

If a column is added to a table on which a mask or a permission is defined, or to a table that is referenced in the definition of a mask or a permission, that mask or permission is invalidated. Access to a table that has column access control activated and an invalid mask defined on it is blocked until the invalid mask is either disabled, dropped, or recreated (SQLSTATE 560D0). Access to a table that has row access control activated and an invalid row permission defined on it is blocked until the invalid permission is either disabled, dropped, or recreated (SQLSTATE 560D0).

column-name
Is the name of the column to be added to the table. The name cannot be qualified. Existing column names or period names in the table cannot be used (SQLSTATE 42711).

data-type
Is one of the data types listed under "CREATE TABLE".

NOT NULL
Prevents the column from containing null values. The *default-clause* must also be specified (SQLSTATE 42601).

lob-options
Specifies options for LOB data types. See *lob-options* in "CREATE TABLE".

SCOPE
Specify a scope for a reference type column.

typed-table-name2
The name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the value actually references an existing row in *typed-table-name2*.

typed-view-name2
The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the values actually references an existing row in *typed-view-name2*.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same ALTER TABLE statement, or as the name of any other existing constraint on the table (SQLSTATE 42710).

If the constraint name is not specified by the user, an 18 byte long identifier unique within the identifiers of the existing constraints defined on the table is generated by the system. (The identifier consists of "SQL" followed by a sequence of 15 numeric characters that are generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint. See Notes for details on index names associated with unique constraints.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in

the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause were specified as a separate clause. The column cannot contain null values, so the NOT NULL attribute must also be specified (SQLSTATE 42831).

See PRIMARY KEY within the *unique-constraint* description.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause were specified as a separate clause.

See UNIQUE within the *unique-constraint* description.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* in "CREATE TABLE".

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See *check-condition* in "CREATE TABLE".

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a specific default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in Table 13. If a column is defined as an XML or structured type, then a DEFAULT clause cannot be specified.

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

Table 13. Default Values (when no value specified)

Data Type	Default Value
Numeric	0
Fixed-length character string	Blanks
Varying-length character string	A string of length 0
Fixed-length graphic string	Double-byte blanks
Varying-length graphic string	A string of length 0
Date	For existing rows, a date corresponding to January 1, 0001. For added rows, the current date.
Time	For existing rows, a time corresponding to 0 hours, 0 minutes, and 0 seconds. For added rows, the current time.

Table 13. Default Values (when no value specified) (continued)

Data Type	Default Value
Timestamp	For existing rows, a date corresponding to January 1, 0001, and a time corresponding to 0 hours, 0 minutes, 0 seconds and 0 microseconds. For added rows, the current timestamp.
Binary string (blob)	A string of length 0

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column.

Specific types of values that can be specified with the DEFAULT keyword are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment as described in Chapter 3
- not be a floating-point constant unless the column is defined with a floating-point data type
- be a numeric constant or a decimal floating-point special value if the data type of the column is decimal floating-point. Floating-point constants are first interpreted as DOUBLE and then converted to decimal floating-point. For DECFLOAT(16) columns, decimal constants must have a precision less than or equal to 16.
- not have nonzero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*.

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified). For existing rows, the value is the current date, current time or current timestamp when the ALTER TABLE statement is processed.

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be

specified in place of CURRENT USER. For existing rows, the value is the CURRENT USER, SESSION_USER, or SYSTEM_USER of the ALTER TABLE statement.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. For existing rows, the value of the CURRENT SCHEMA special register at the time the ALTER TABLE statement is processed.

CURRENT MEMBER
Specifies the value of the CURRENT MEMBER special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT MEMBER is specified, the data type of the column must allow assignment from an integer. For existing rows, the value of the CURRENT MEMBER special register at the time the ALTER TABLE statement is processed.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL must not be specified within the same column definition.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

EMPTY_CLOB(), EMPTY_DBCLOB(), or EMPTY_BLOB()
Specifies a zero-length string as the default for the column. The column must have the data type that corresponds to the result data type of the function.

If the value specified is not valid, an error (SQLSTATE 42894) is returned.

generated-clause
Specifies a generated value for the column. This clause must not be specified with *default-clause* in a column definition (SQLSTATE 42623). A generated column cannot be added to a system-period temporal table (SQLSTATE 428HZ). For details on column generation, see "CREATE TABLE".

GENERATED

Specifies that the database manager generates values for the column. GENERATED must be specified if the column is to be considered an identity column, row change timestamp column, row-begin column, row-end column, transaction start-ID column, or generated expression column.

If the column is nullable, the null value is assigned as the value for the column in existing rows. Otherwise, the value for the column in existing rows depends on the definition of the column:

- ROW CHANGE TIMESTAMP uses a value that corresponds to the timestamp of the ALTER TABLE statement
- ROW BEGIN uses a date that corresponds to January 1, 0001 and a time that corresponds to 0 hours, 0 minutes, 0 seconds, and 0 fractional seconds
- ROW END uses a date that corresponds to December 30, 9999, and a time that corresponds to 0 hours, 0 minutes, 0 seconds, and 0 fractional seconds
- TRANSACTION START ID uses a date that corresponds to January 1, 0001, and a time that corresponds to 0 hours, 0 minutes, 0 seconds, and 0 fractional seconds
- Expressions use the value derived from the expression

ALWAYS
Specifies that the database manager will always generate a value for the column when a row is inserted or updated and a value must be generated. The result of the expression is stored in the

table. GENERATED ALWAYS is the recommended option unless data propagation or unload and reload operations are being performed. GENERATED ALWAYS is the default for generated columns.

BY DEFAULT
Specifies that the database manager will generate a value for the column when a row is inserted into the table, or updated, specifying DEFAULT for the column, unless an explicit value is specified. BY DEFAULT can only be specified with *as-row-change-timestamp-clause*. BY DEFAULT is the recommended option when using data propagation or performing unload and reload operations.

FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
Specifies that the column is a timestamp column with values generated by the database manager. A value is generated for the column in each row that is inserted, and for any row in which any column is updated. The value that is generated for a ROW CHANGE TIMESTAMP column is a timestamp that corresponds to the insert or update time for that row. If multiple rows are inserted or updated with a single statement, the value of the ROW CHANGE TIMESTAMP column might be different for each row.

A table can only have one ROW CHANGE TIMESTAMP column (SQLSTATE 428C1). If *data-type* is specified, it must be TIMESTAMP or TIMESTAMP(6) (SQLSTATE 42842). A ROW CHANGE TIMESTAMP column cannot have a DEFAULT clause (SQLSTATE 42623). NOT NULL must be specified for a ROW CHANGE TIMESTAMP column (SQLSTATE 42831).

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. Requires that the table be put in set integrity pending no access state, using the SET INTEGRITY statement with the OFF NO ACCESS option. After the ALTER TABLE statement, the SET INTEGRITY statement with the IMMEDIATE CHECKED and FORCE GENERATED options must be used to update and check all the values in that column against the new expression. For details on specifying a column with a *generation-expression*, see "CREATE TABLE".

AS ROW BEGIN

Specifies that the value is assigned by the database manager whenever a row is inserted into the table or any column in the row is updated. The value is generated using a reading of the time-of-day clock during execution of the first of the following events in the transaction:

- A data change statement that requires a value to be assigned to the row-begin or transaction start-ID column in a table
- A deletion of a row in a system-period temporal table

For a system-period temporal table, the database manager ensures uniqueness of the generated values for a row-begin column across transactions. The timestamp value might be adjusted to ensure that rows inserted into an associated history table have the end timestamp value greater than the begin timestamp value (SQLSTATE 01695). This can happen when a conflicting transaction

is updating the same row in the system-period temporal table. The database configuration parameter **systime_period_adj** must be set to Yes for this adjustment to the timestamp value to occur otherwise an error is returned (SQLSTATE 57062). If multiple rows are inserted or updated within a single SQL transaction and an adjustment is not needed, the values for the row-begin column are the same for all the rows and are unique from the values generated for the column for another transaction. A row-begin column is required as the begin column of a SYSTEM_TIME period, which is the intended use for this type of generated column.

A table can have only one row-begin column (SQLSTATE 428C1). If *data-type* is not specified the column is defined as a TIMESTAMP(12). If *data-type* is specified, it must be TIMESTAMP(12) (SQLSTATE 42842). The column must be defined as NOT NULL (SQLSTATE 42831). A row-begin column is not updatable.

AS ROW END

Specifies that the maximum value for the data type of the column is assigned by the database manager whenever a row is inserted or any column in the row is updated.

A row-end column is required as the second column of a SYSTEM_TIME period, which is the intended use for this type of generated column.

A table can have only one row-end column (SQLSTATE 428C1). If *data-type* is not specified, the column is defined as TIMESTAMP(12). If *data-type* is specified, it must be TIMESTAMP(12) (SQLSTATE 42842). The column must be defined as NOT NULL (SQLSTATE 42831). A row-end column is not updatable.

AS TRANSACTION START ID

Specifies that the value is assigned by the database manager whenever a row is inserted into the table or any column in the row is updated. The database manager assigns a unique timestamp value per transaction or the null value. The null value is assigned to the transaction start-ID column if the column is nullable and if there is a row-begin column in the table for which the value did not need to be adjusted. Otherwise the value is generated using a reading of the time-of-day clock during execution of the first of the following events in the transaction:

- A data change statement that requires a value to be assigned to the row-begin or transaction start-ID column in a table
- A deletion of a row in a system-period temporal table

If multiple rows are inserted or updated within a single SQL transaction, the values for the transaction start-ID column are the same for all the rows and are unique from the values generated for the column for another transaction.

A transaction start-ID column is required for a system-period temporal table, which is the intended use for this type of generated column.

A table can have only one transaction start-ID column (SQLSTATE 428C1). If *data-type* is not specified, the column is defined as TIMESTAMP(12). If *data-type* is specified it must be TIMESTAMP(12). A transaction start-ID column is not updatable.

COMPRESS SYSTEM DEFAULT
Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned (SQLSTATE 01648) and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be a DATE, TIME, TIMESTAMP, XML, or structured data type (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

COLUMN SECURED WITH *security-label-name*
Identifies a security label that exists for the security policy that is associated with the table. The name must not be qualified (SQLSTATE 42601). The table must have a security policy associated with it (SQLSTATE 55064). The table must not be a system-period temporal table.

NOT HIDDEN or IMPLICITLY HIDDEN
Specifies whether the column is to be defined as hidden. The hidden attribute determines whether the column is included in an implicit reference to the table, or whether it can be explicitly referenced in SQL statements. The default is NOT HIDDEN.

NOT HIDDEN
Specifies that the column is included in implicit references to the table, and that the column can be explicitly referenced.

IMPLICITLY HIDDEN
Specifies that the column is not visible in SQL statements unless the column is explicitly referenced by name. For example, assuming that a table includes a column defined with the IMPLICITLY HIDDEN clause, the result of a `SELECT *` does not include the implicitly hidden column. However, the result of a SELECT that explicitly refers to the name of an implicitly hidden column will include that column in the result table.

ADD *unique-constraint*
Defines a unique or primary key constraint. A primary key or unique constraint cannot be added to a table that is a subtable (SQLSTATE 429B3). If the table is a supertable at the top of the hierarchy, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*
Names the primary key or unique constraint. For more information, see *constraint-name* in "CREATE TABLE".

UNIQUE (*column-name***, ... BUSINESS_TIME WITHOUT OVERLAPS)**
Defines a unique key composed of the identified columns and periods. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified

more than once. The name cannot be qualified. The number of identified columns plus two times the number of identified periods must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL and XML limits". No LOB, distinct type based on any of these types, or structured type can be used as part of a unique key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). The set of columns in the unique key cannot be the same as the set of columns of the primary key or another unique key (SQLSTATE 01543). If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891. Any existing values in the set of identified columns must be unique (SQLSTATE 23515).

A check is performed to determine whether an existing index matches the unique key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. However, for partitioned tables, non-unique partitioned indexes whose columns are not a superset of the table-partitioning key columns are not considered matching indexes.

When a partition is attached to a range partitioned application-period temporal table that has a partitioned BUSINESS_TIME WITHOUT OVERLAPS index, the source table must have an index that matches the partitioned BUSINESS_TIME WITHOUT OVERLAPS index. Additionally, the PERIODNAME and PERIODPOLICY attributes on the indexes must also match.

If a matching index definition is found, the description of the index is changed to indicate that it is required by the system and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected. If there are multiple unique indexes, the selection is arbitrary with one exception:

- For partitioned tables, matching unique partitioned indexes are favored over matching unique nonpartitioned indexes or matching non-unique indexes (partitioned or nonpartitioned).

If no matching index is found, a unique bidirectional index will automatically be created for the columns, as described in CREATE TABLE. See Notes for details on index names associated with unique constraints.

BUSINESS_TIME WITHOUT OVERLAPS
For a constraint, BUSINESS_TIME indicates the period name in this table. The period must exist in the table (SQLSTATE 42727).

BUSINESS_TIME WITHOUT OVERLAPS specifies that overlapping periods for BUSINESS_TIME are not allowed, and that values for the rest of the keys must be unique with respect to any period of BUSINESS_TIME. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the end column and begin column of the period BUSINESS_TIME (in this order of the columns) will automatically be added to the index key in ascending order and enforce that there are no overlaps in time. The columns used to defined BUSINESS_TIME must not be specified as part of the constraint (SQLSTATE 428HW).

PRIMARY KEY (*column-name*, ... BUSINESS_TIME WITHOUT OVERLAPS)
Defines a primary key composed of the identified columns. Each

column-name must identify a column of the table, and the same column must not be identified more than once. The name cannot be qualified. The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL limits". The table must not have a primary key and the identified columns must be defined as NOT NULL. No LOB, distinct type based on any of these types, or structured type may be used as part of a primary key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.) Any existing values in the set of identified columns must be unique (SQLSTATE 23515). *column-name* must not be the name of a row change timestamp, or a begin or end column of the period (SQLSTATE 428HW).

A check is performed to determine if an existing index matches the primary key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. However, for partitioned tables, non-unique partitioned indexes whose columns are not a superset of the table-partitioning key columns are not considered matching indexes.

When a partition is attached to a range partitioned application-period temporal table that has a partitioned BUSINESS_TIME WITHOUT OVERLAPS index, the source table must have an index that matches the partitioned BUSINESS_TIME WITHOUT OVERLAPS index. Additionally, the PERIODNAME and PERIODPOLICY attributes on the indexes must also match.

If a matching index definition is found, the description of the index is changed to indicate that it is the primary index, as required by the system, and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected. If there are multiple unique indexes, the selection is arbitrary with one exception:

- For partitioned tables, matching unique partitioned indexes are favored over matching unique nonpartitioned indexes or matching non-unique indexes (partitioned or nonpartitioned).

If no matching index is found, a unique bidirectional index will automatically be created for the columns, as described in CREATE TABLE. See Notes for details on index names associated with unique constraints.

Only one primary key can be defined on a table.

BUSINESS_TIME WITHOUT OVERLAPS

For a constraint, BUSINESS_TIME indicates the period name in this table. The period must exist in the table (SQLSTATE 42727).

BUSINESS_TIME WITHOUT OVERLAPS specifies that overlapping periods for BUSINESS_TIME are not allowed, and that values for the rest of the keys must be unique with respect to any period of BUSINESS_TIME. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the end column and begin column of the period BUSINESS_TIME (in this order of the columns) will automatically be added to the index key in ascending order and enforce that there are

no overlaps in time. The columns used to defined BUSINESS_TIME must not be specified as part of the constraint (SQLSTATE 428HW).

ADD *referential-constraint*
Defines a referential constraint. See *referential-constraint* in "CREATE TABLE".

ADD *check-constraint*
Defines a check constraint or functional dependency. See *check-constraint* in "CREATE TABLE".

ADD *distribution-clause*
Defines a distribution key. The table must be defined in a table space on a single-partition database partition group (SQLSTATE 55037) and must not already have a distribution key (SQLSTATE 42889). If a distribution key already exists for the table, the existing key must be dropped before adding the new distribution key. A distribution key cannot be added to a table that is a subtable (SQLSTATE 428DH) .

DISTRIBUTE BY HASH (*column-name***...)**
Defines a distribution key using the specified columns. Each *column-name* must identify a column of the table, and the same column must not be identified more than once. The name cannot be qualified. A column cannot be used as part of a distribution key if the data type of the column is a BLOB, CLOB, DBCLOB, XML, distinct type on any of these types, or structured type.

ADD RESTRICT ON DROP
Specifies that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

ADD MATERIALIZED QUERY

materialized-query-definition
Changes a regular table to a materialized query table for use during query optimization. The table specified by *table-name* must not:

- Be previously defined as a materialized query table
- Be a typed table
- Have any constraints, unique indexes, or triggers defined
- Reference a nickname that is marked with caching disabled
- Be referenced in the definition of another materialized query table
- Be referenced in the definition of a view that is enabled for query optimization

If *table-name* does not meet these criteria, an error is returned (SQLSTATE 428EW).

If row level or column level access control is activated for any table that is directly or indirectly referenced in the *fullselect* of *materizalized-query-definition*, and row level access control is not activated for the table being altered, row level access control is implicitly activated for the altered table. This restricts direct access to the contents of the materialized query table. A query that explicitly references the table before such a row permission is defined returns a warning that there is no data in the table (SQLSTATE 02000). To provide access to the materialized query table, an appropriate row permission can be created, or an **ALTER TABLE DEACTIVATE ROW ACCESS CONTROL** on the materialized query table can be issued to remove the row level protection if that is appropriate.

If the materialized query table references any table that has row level or column level access control activated, the functions referenced in the *fullselect* of *materizalized-query-definition* must be defined with the SECURED attribute (SQLSTATE 428EC).

If the table being altered to a materialized query table has any permissions (excluding the system generated default permission) or masks defined on it, ALTER fails (SQLSTATE 428EW).

fullselect
Defines the query in which the table is based. The columns of the existing table must:

- have the same number of columns
- have exactly the same data types
- have the same column names in the same ordinal positions

as the result columns of *fullselect* (SQLSTATE 428EW). For details about specifying the *fullselect* for a materialized query table, see "CREATE TABLE". One additional restriction is that *table-name* cannot be directly or indirectly referenced in the fullselect.

refreshable-table-options
Specifies the refreshable options for altering a materialized query table.

DATA INITIALLY DEFERRED
The data in the table must be validated using the REFRESH TABLE or SET INTEGRITY statement.

REFRESH
Indicates how the data in the table is maintained.

DEFERRED
The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

IMMEDIATE
The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified subselect is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION
The materialized query table can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY
Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool.

SYSTEM
Specifies that the data in the materialized query table is maintained by the system.

USER
Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL
Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

ALTER FOREIGN KEY *constraint-name*
Alters the constraint attributes of the referential constraint *constraint-name*. The *constraint-name* must identify an existing referential constraint (SQLSTATE 42704).

ALTER CHECK *constraint-name*
Alters the constraint attributes of the check constraint or functional dependency *constraint-name*. The *constraint-name* must identify an existing check constraint or functional dependency (SQLSTATE 42704).

constraint-alteration
Options for changing attributes associated with referential or check constraints.

ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION
Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances.

ENABLE QUERY OPTIMIZATION
The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

ENFORCED or NOT ENFORCED
Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete.

ENFORCED
Change the constraint to ENFORCED. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621).

NOT ENFORCED
Change the constraint to NOT ENFORCED.

TRUSTED
The data can be trusted to conform to the constraint. TRUSTED must be used only if the data in the table is independently known

to conform to the constraint. Query results might be unpredictable if the data does not actually conform to the constraint. This is the default option.

Informational constraints must not be violated at any time. Informational constraints are used in query optimization, as well as the incremental processing of REFRESH IMMEDIATE MQT and staging tables. These processes might produce unpredictable results or incorrect MQT and staging table content if the constraints are violated. For example, the order in which parent-child tables are maintained is important. When you want to add rows to a parent-child table, you must insert rows into the parent table first. To remove rows from a parent-child table, you must delete rows from the child table first. This ensures that there are no orphan rows in the child table at any time. If informational constraints are violated, the incremental maintenance of dependent MQT data and staging table data might be optimized based on the violated informational constraints, producing incorrect data.

NOT TRUSTED
The data cannot be trusted to conform to the constraint. NOT TRUSTED is intended for cases where the data conforms to the constraint for most rows, but it is not independently known that all the rows or future additions will conform to the constraint. If a constraint is NOT TRUSTED and enabled for query optimization, then it will not be used to perform optimizations that depend on the data conforming completely to the constraint. NOT TRUSTED can be specified only for referential integrity constraints (SQLSTATE 42613).

ALTER *column-alteration*
Alters the definition of a column. Only the specified attributes will be altered; others will remain unchanged. Columns of a typed table cannot be altered (SQLSTATE 428DH). The table must not be defined as a history table (SQLSTATE 428FR).

column-name
Specifies the name of the column that is to be altered. The *column-name* must identify an existing column of the table (SQLSTATE 42703). The name must not be qualified. The name must not identify a column that is otherwise being added, altered, or dropped in the same ALTER TABLE statement (SQLSTATE 42711).

SET DATA TYPE *altered-data-type*
Specifies the new data type of the column. The new data type must be castable from the existing data type of the column (SQLSTATE 42837) except when one of the data types is a distinct type, in which case the source data type of the distinct type is used in determining if the data types are castable.

Altering a string data type that results in the truncation of non-blank characters from existing data is not allowed (SQLSTATE 42837).

Data type alterations require a table reorganization before the table can be fully accessed (SQLSTATE 57016), except in the following situations:

- Increasing the length of a VARCHAR or VARGRAPHIC column
- Decreasing the length of a VARCHAR or VARGRAPHIC column without truncating trailing blanks from existing data

The administrative routine SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS can be called to do table reorganization as required. A data type alteration that requires a table reorganization cannot be specified if the table is in SET INTEGRITY PENDING state (SQLSTATE 57007).

A string data type cannot be altered if the column is a column of a table-partitioning key.

If the column is a column of a distribution key, then the new data type must meet the following requirements (SQLSTATE 42997):

- Be the same data type as the current column type
- Have the same length of the current column type, except in the case of increasing column length of VARCHAR and VARGRAPHIC data type columns
- Cannot be modified to FOR BIT DATA or vice-versa in the cases of CHAR and VARCHAR data types

The specified length cannot be less than the existing length if the data type is a LOB (SQLSTATE 42837).

The data type of an identity column cannot be altered (SQLSTATE 42997).

The data type of a column defined as ROW BEGIN, ROW END, or TRANSACTION START ID cannot be altered (SQLSTATE 428FR).

The data type and nullability of BUSINESS_TIME period columns cannot be altered (SQLSTATE 428FR).

The table cannot have data capture enabled (SQLSTATE 42997).

The data type of a column cannot be altered if any of the following conditions are true (SQLSTATE 42893):

- The column is a generated expression column and the data of the generated expression column will change if the column is altered
- The column is referenced in an expression of a generated expression column and the data of the generated expression column will change if the column is altered
- The column is referenced in a check constraint and the check constraint will not be satisfied if the column is altered
- The column is used in a referential integrity constraint and the referential integrity constraint will not be satisfied if the column is altered

Altering a column must not make the total byte count of all columns exceed the maximum record size (SQLSTATE 54010). If the column is used in a unique constraint or an index, the new length must not cause the sum of the stored lengths for the unique constraint or index to exceed the index key length limit for the page size (SQLSTATE 54008). For column stored lengths, see “Byte Counts” in “CREATE TABLE”. For key length limits, see “SQL and XML limits”.

If **auto_reval** is set to DISABLED, the cascaded effects of altering a column is shown in Table 14 on page 141.

If either a row permission or a column mask is dependent on the column being altered (as recorded in the SYSCAT.CONTROLDEP catalog view), an error is returned (SQLSTATE 42917).

Table 14. Cascaded effects of altering a column

Operation	Effect
Altering a column that is referenced by a view or check constraint	The object is regenerated during alter processing. In the case of a view, function or method resolution for the object might be different after the alter operation, changing the semantics of the object. In the case of a check constraint, if the semantics of the object will change as a result of the alter operation, the operation fails.
Altering a column in a table that has a dependent package, trigger, or SQL routine	The object is marked invalid, and is revalidated on next use.
Altering the type of a column in a table that is referenced by an XSROBJECT enabled for decomposition	The XSROBJECT is marked inoperative for decomposition. Re-enabling the XSROBJECT might require readjustment of its mappings; following this, issue an ALTER XSROBJECT ENABLE DECOMPOSITION statement against the XSROBJECT.
Altering a column that is referenced in the default expression of a global variable	The default expression of the global variable is validated during alter processing. If a user-defined function used in the default expression cannot be resolved, the operation fails.

If the table is a system-period temporal table, the column is also changed in any associated history table. If the table is a system-period temporal table, string data type columns cannot be altered to a length that requires data truncation, and numeric data type columns cannot be altered to lower precision data types (SQLSTATE 42837).

SET NOT NULL
Specifies that the column cannot contain null values. No value for this column in existing rows of the table can be the null value (SQLSTATE 23502). This clause is not allowed if the column is specified in the foreign key of a referential constraint with a DELETE rule of SET NULL, and no other nullable columns exist in the foreign key (SQLSTATE 42831). Altering this attribute for a column requires table reorganization before further table access is allowed (SQLSTATE 57016). Note that because this operation requires validation of table data, it cannot be performed when the table is in reorg pending state (SQLSTATE 57016). The table cannot have data capture enabled (SQLSTATE 42997).

If a row permission or column mask exists, which depends on the column to be altered, an error will be issued (SQLSTATE 42917).

If the table is a system-period temporal table, the column is also changed in any associated history table.

SET INLINE LENGTH *integer*
Changes the inline length of an existing structured type, XML, or LOB data type column. The inline length indicates the maximum size in bytes of an instance of a structured type, XML, or LOB data type to store in the base table row. Instances of a structured type or XML data type that cannot be stored inline in the base table row are stored separately, similar to the way that LOB values are stored.

The data type of *column-name* must be a structured type, XML, or LOB data type (SQLSTATE 42842).

The default inline length for a structured type column is the inline length of its data type (specified explicitly or by default in the CREATE TYPE statement). If the inline length of a structured type is less than 292, the value 292 is used for the inline length of the column.

The explicit inline length value can only be increased (SQLSTATE 429B2); it cannot exceed 32673 (SQLSTATE 54010). For a structured type or XML data type column, it must be at least 292. For a LOB data type column, the INLINE LENGTH must not be less than the maximum LOB descriptor size.

Altering the column must not make the total byte count of all columns exceed the row size limit (SQLSTATE 54010).

Data that is already stored separately from the rest of the row will not be moved inline into the base table row by this statement. To take advantage of the altered inline length of a structured type column, invoke the REORG command against the specified table after altering the inline length of its column. To take advantage of the altered inline length of an XML data type column in an existing table, update all rows with an UPDATE statement. The REORG command has no effect on the row storage of XML documents. To take advantage of the altered inline length of a LOB data type column, use the REORG command with the LONGLOBDATA option or UPDATE the corresponding LOB column. For example:

```
UPDATE table-name SET lob-column = lob-column
   WHERE LENGTH(lob-column) <= chosen-inline-length - 4
```

where *table-name* is the table that had the inline length of the LOB data type column altered, *lob-column* is the LOB data type column that was altered, and *chosen-inline-length* is the new value that was chosen for the INLINE LENGTH.

If a row permission or column mask exists, which depends on the column to be altered, an error will be returned (SQLSTATE 42917).

If the table is a system-period temporal table, inline length changes are propagated to the history table.

SET *default-clause*
Specifies a new default value for the column that is to be altered. The column must not already be defined as a generated column (SQLSTATE 42623). The specified default value must represent a value that could be assigned to the column in accordance with the rules for assignment as described in "Assignments and comparisons". Altering the default value does not change the value that is associated with this column for existing rows.

SET EXPRESSION AS (*generation-expression***)**
Changes the expression for the column to the specified *generation-expression*. SET EXPRESSION requires the table to be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option. After the ALTER TABLE statement, the SET INTEGRITY statement with the IMMEDIATE CHECKED and FORCE GENERATED options must be used to update and check all the values in that column against the new expression. The column must already be defined as a generated column based on an expression (SQLSTATE 42837), and must not have appeared in the PARTITIONING KEY, DIMENSIONS, or KEY SEQUENCE clauses of the table (SQLSTATE 42997). The generation-expression must conform to

the same rules that apply when defining a generated column. The result data type of the generation-expression must be assignable to the data type of the column (SQLSTATE 42821).

The *generation-expression* must not reference a column for which a column mask is defined (SQLSTATE 42621).

SET NOT HIDDEN or SET IMPLICITLY HIDDEN
Specifies the hidden attribute for the column.

If the table is a system-period temporal table, the column is also changed in any associated history table.

NOT HIDDEN
Specifies that the column is included in implicit references to the table, and that the column can be explicitly referenced.

IMPLICITLY HIDDEN
Specifies that the column is not visible in SQL statements unless the column is explicitly referenced by name. For example, assuming that a table includes a column defined with the IMPLICITLY HIDDEN clause, the result of a `SELECT *` does not include the implicitly hidden column. However, the result of a SELECT that explicitly refers to the name of an implicitly hidden column will include that column in the result table.

IMPLICITLY HIDDEN must not be specified for the last column of the table that is not hidden (SQLSTATE 428GU).

SET *generation-alteration*
Specifies that the generation attribute for the column is to be changed. GENERATED may be specified if the column is an identity column or a row change timestamp column (SQLSTATE 42837). If the table is a system-period temporal table, the column in the associated history table is not affected by the change. If there is an existing default for the column, that default must be dropped, which can be done in the same *column-alteration* using one of the DROP DEFAULT clause. SET GENERATED must not be specified for a column of a temporal history table (SQLSTATE 428FR).

GENERATED ALWAYS
Specifies that the database manager will always generate a value for the column when a row is inserted or updated and a value must be generated. GENERATED ALWAYS is the recommended option unless data propagation or unload and reload operations are being performed. ALWAYS is the default for generated columns.

GENERATED BY DEFAULT
Specifies that the database manager will generate a value for the column when a row is inserted into the table, or updated, specifying DEFAULT for the column, unless an explicit value is specified. GENERATED BY DEFAULT can only be specified with *as-row-change-timestamp-clause*. GENERATED BY DEFAULT is the recommended option when using data propagation or performing unload and reload operations.

identity-alteration
Alters the identity attributes of the column. The column must be an identity column.

SET INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the identity column. The next value to be generated for the identity column will be determined from the last assigned value with the increment applied. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence after the ALTER statement. If this value is 0 or positive, this is an ascending sequence after the ALTER statement.

SET NO MINVALUE or MINVALUE *numeric-constant*
Specifies the minimum value at which a descending identity column either cycles or stops generating values, or the value to which an ascending identity column cycles after reaching the maximum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MINVALUE
For an ascending sequence, the value is the original starting value. For a descending sequence, the value is the minimum value of the data type of the column.

MINVALUE *numeric-constant*
Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

SET NO MAXVALUE or MAXVALUE *numeric-constant*
Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or the value to which a descending identity column cycles after reaching the minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MAXVALUE
For an ascending sequence, the value is the maximum value of the data type of the column. For a descending sequence, the value is the original starting value.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

SET NO CYCLE or CYCLE
Specifies whether this identity column should continue to generate

values after generating either its maximum or minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CYCLE
Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached.

CYCLE
Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, then after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, duplicate values can be generated for an identity column. Although not required, if unique values are desired, a single-column unique index defined using the identity column will ensure uniqueness. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

SET NO CACHE or CACHE *integer-constant*
Specifies whether to keep some pre-allocated values in memory for faster access. This is a performance and tuning option. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CACHE
Specifies that values for the identity column are not to be pre-allocated. In a DB2 pureScale environment, if the identity values *must* be generated in order of request, the NO CACHE option must be used.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*
Specifies how many values of the identity sequence are pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value requires waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used in committed statements are lost (that is, they will never be used). The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of system failure.

The minimum value is 2 (SQLSTATE 42815).

In a DB2 pureScale environment, if both CACHE and ORDER are specified, the specification of ORDER overrides the specification of CACHE and instead NO CACHE will be in effect.

SET NO ORDER or ORDER
Specifies whether the identity column values must be generated in order of request. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO ORDER
Specifies that the identity column values do not need to be generated in order of request.

ORDER
Specifies that the identity column values must be generated in order of request.

RESTART or RESTART WITH *numeric-constant*
Resets the state of the sequence associated with the identity column. If WITH *numeric-constant* is not specified, the sequence for the identity column is restarted at the value that was specified, either implicitly or explicitly, as the starting value when the identity column was originally created.

The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837). RESTART does *not* change the original START WITH value.

The *numeric-constant* is an exact numeric constant that can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA). The *numeric-constant* will be used as the next value for the column.

SET *generation-attribute as-identity-clause*
Changes the column to an identity column. This column alteration must not be specified if the column has a default or is already a generated column (SQLSTATE 42837). If the table is a system-period temporal table, the column in the associated history table is not affected by the change.

GENERATED ALWAYS
Specifies that the database manager will always generate a value for the column when a row is inserted or updated and a value must be generated. ALWAYS is the default for generated columns.

GENERATED BY DEFAULT
Specifies that the database manager generates a value for the column when a row is inserted or updated and a default value must be generated, unless an explicit value is specified.

as-identity-clause
Specifies that the column is the identity column for the table. A table can only have a single identity column (SQLSTATE 428C1). The column must be specified as not nullable (SQLSTATE 42997), and the data type associated with the column must be an exact numeric data type with a scale of zero (SQLSTATE 42815). An exact numeric data type is one of: SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC with a scale of zero, or a distinct type based on one of these types. For details on identity options, see “CREATE TABLE”.

SET GENERATED ALWAYS
Changes the column to a generated expression column, a row-begin column, a row-end column, or a transaction-start-ID column. GENERATED ALWAYS specifies that the database manager will always generate a value for the column when a row is inserted or updated and a value must be generated.

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. The column must not already be defined with a generation expression, cannot be the identity column, or cannot have an explicit default (SQLSTATE 42837). The *generation-expression* must conform to the same rules that apply when defining a generated column. The result data type of the *generation-expression* must be assignable to the data type of the column (SQLSTATE 42821). The column must not be referenced in the distribution key column or in the ORGANIZE BY clause (SQLSTATE 42997).

The *generation-expression* must not reference a column for which a column mask is defined (SQLSTATE 42621).

AS ROW BEGIN

Specifies that the value is assigned by the database manager whenever a row is inserted into the table or any column in the row is updated. The value is generated using a reading of the time-of-day clock during execution of the first of the following events in the transaction:

- A data change statement that requires a value to be assigned to the row-begin or transaction start-ID column in a table
- A deletion of a row in a system-period temporal table

For a system-period temporal table, the database manager ensures uniqueness of the generated values for a row-begin column across transactions. The timestamp value might be adjusted to ensure that rows inserted into an associated history table have the end timestamp value greater than the begin timestamp value (SQLSTATE 01695). This can happen when a conflicting transaction is updating the same row in the system-period temporal table. The database configuration parameter **systime_period_adj** must be set to `Yes` for this adjustment to the timestamp value to occur otherwise an error is returned (SQLSTATE 57062). If multiple rows are inserted or updated within a single SQL transaction and an adjustment is not needed, the values for the row-begin column are the same for all the rows and are unique from the values generated for the column for another transaction. A row-begin column is required as the begin column of a SYSTEM_TIME period, which is the intended use for this type of generated column.

A table can have only one row-begin column (SQLSTATE 428C1). If *data-type* is not specified the column is defined as a TIMESTAMP(12). If *data-type* is specified, it must be TIMESTAMP(12) (SQLSTATE 42842). The column must be defined as NOT NULL (SQLSTATE 42831). A row-begin column is not updatable.

AS ROW END

Specifies that the maximum value for the data type of the column is assigned by the database manager whenever a row is inserted or any column in the row is updated.

A row-end column is required as the second column of a SYSTEM_TIME period, which is the intended use for this type of generated column.

A table can have only one row-end column (SQLSTATE 428C1). If *data-type* is not specified, the column is defined as TIMESTAMP(12). If *data-type* is specified, it must be TIMESTAMP(12) (SQLSTATE 42842). The column must be defined as NOT NULL (SQLSTATE 42831). A row-end column is not updatable.

AS TRANSACTION START ID

Specifies that the value is assigned by the database manager whenever a row is inserted into the table or any column in the row is updated. The database manager assigns a unique timestamp value per transaction or the null value. The null value is assigned to the transaction start-ID column if the column is nullable and if there is a row-begin column in the table for which the value did not need to be adjusted. Otherwise the value is generated using a reading of the time-of-day clock during execution of the first of the following events in the transaction:

- A data change statement that requires a value to be assigned to the row-begin or transaction start-ID column in a table
- A deletion of a row in a system-period temporal table

If multiple rows are inserted or updated within a single SQL transaction, the values for the transaction start-ID column are the same for all the rows and are unique from the values generated for the column for another transaction.

A transaction start-ID column is required for a system-period temporal table, which is the intended use for this type of generated column.

A table can have only one transaction start-ID column (SQLSTATE 428C1). If *data-type* is not specified, the column is defined as TIMESTAMP(12). If *data-type* is specified it must be TIMESTAMP(12). A transaction start-ID column is not updatable.

DROP DEFAULT

Drops the current default for the column. The specified column must have a default value (SQLSTATE 42837). This action is propagated to the history table for a system-period temporal table.

DROP GENERATED

Drops the generated attributes of the column. The column must be defined as a generated column (SQLSTATE 42837). The column must not be defined as a row-begin column, row-end column, or a transaction-start-ID column in a system-period temporal table (SQLSTATE 428FR).

DROP NOT NULL

Drops the NOT NULL attribute of the column, allowing the column to have the null value. This clause is not allowed if the column is specified in the primary key, in a unique constraint of the table (SQLSTATE 42831), a row-begin column, or a row-end column (SQLSTATE 42837). Altering this attribute for a column requires table reorganization before further table access is allowed (SQLSTATE 57016). The table cannot have data capture enabled (SQLSTATE 42997). DROP NOT NULL is blocked for columns belonging to the BUSINESS_TIME period (SQLSTATE 428FR).

If the table is a system-period temporal table, the NOT NULL attribute is also dropped from the corresponding column in any associated history table.

If either a row permission or column mask exists, which depends on the column to be altered, an error will be issued (SQLSTATE 42917).

ADD SCOPE
Add a scope to an existing reference type column that does not already have a scope defined (SQLSTATE 428DK). If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

typed-table-name
The name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

COMPRESS
Specifies whether or not default values for this column are to be stored more efficiently.

SYSTEM DEFAULT
Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the table is not already set with the VALUE COMPRESSION attribute activated, a warning is returned (SQLSTATE 01648), and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of the extra checking that is done.

Existing data in the column is not changed. Consider offline table reorganization to enable existing data to take advantage of storing system default values using minimal space.

OFF
Specifies that system default values are to be stored in the column as regular values. Existing data in the column is not changed. Offline reorganization is recommended to change existing data.

The base data type must not be DATE, TIME or TIMESTAMP (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

SECURED WITH *security-label-name*
Identifies a security label that exists for the security policy that is associated with the table. The name must not be qualified (SQLSTATE

42601). The table must have a security policy associated with it (SQLSTATE 55064). The table must not be a system-period temporal table.

DROP COLUMN SECURITY
Alters a column to make it a non-protected column.

ACTIVATE ROW ACCESS CONTROL

Activates row level access control on the table. The table must not be a typed table, a catalog table (SQLSTATE 55019), a created temporary table, a declared temporary table (SQLSTATE 42995), a nickname (SQLSTATE 42809), or a view (SQLSTATE 42809).

A default row permission is implicitly created and allows no access to any rows of the table, unless permitted by a row permission explicitly created by a user with SECADM authority.

When the table is referenced in a data manipulation statement, all enabled row permissions that have been created for the table, including the default row permission, are applied implicitly by the DB2 database to control the set of rows in the table that are accessible.

If a trigger exists for the table, the trigger must be defined with the SECURED attribute (SQLSTATE 55019).

The table must not be referenced in the definition of a view if an INSTEAD OF trigger that is defined with the NOT SECURED attribute exists for the view (SQLSTATE 55019).

If a materialized query table references the table, the functions referenced in the *fullselect* of *materizalized-query-definition* must be defined with the SECURED attribute (SQLSTATE 55019).

If a materialized query table (or a staging table) that depends on the table (directly or indirectly through a view) for which row level access control is being activated and that materialized query table (or a staging table) does not already have row level access control activated, row level access control is implicitly activated for the materialized query table (or a staging table). This restricts direct access to the contents of the materialized query table (or a staging table). A query that explicitly references the table before such a row permission is defined will return a warning that there is no data in the table (SQLSTATE 02000). To provide access to the materialized query table (or a staging table), an appropriate row permission can be created, or an ALTER TABLE DEACTIVATE ROW ACCESS CONTROL statement on the materialized query table (or a staging table) can be issued to remove the row level protection if that is appropriate.

ACTIVATE ROW ACCESS CONTROL is ignored if row access control is already defined as activated for the table.

If the table is a system-period temporal table, the database manager automatically activates row access control on the history table and creates a default row permission for the history table.

ACTIVATE COLUMN ACCESS CONTROL

Activates column level access control on the table. The table must not be a typed table, a catalog table (SQLSTATE 55019), a created temporary table, a declared temporary table (SQLSTATE 42995), a nickname (SQLSTATE 42809) or a view (SQLSTATE 42809).

The access to the table is not restricted but when the table is referenced in a data manipulation statement, all enabled column masks that have been created

for the table are applied implicitly by the database manager to mask the values returned for the columns referenced in the final result table of the queries.

If a trigger exists for the table, the trigger must be defined with the SECURED attribute (SQLSTATE 55019).

If a materialized query table references the table, the functions referenced in the *fullselect* of *materizalized-query-definition* must be defined with the SECURED attribute (SQLSTATE 55019).

The table must not be referenced in the definition of a view if an INSTEAD OF trigger that is defined with the NOT SECURED attribute exists for the view (SQLSTATE 55019). If a materialized query table that depends on the table (directly or indirectly through a view) for which column level access control is being activated and that materialized query table does not already have row level access control activated, row level access control is implicitly activated for the materialized query table. This restricts direct access to the contents of the materialized query table. A query that explicitly references the table before such a row permission is defined returns a warning that there is no data in the table (SQLSTATE 02000). To provide access to the materialized query table, an appropriate row permission can be created, or an ALTER TABLE DEACTIVATE ROW ACCESS CONTROL statement on the materialized query table can be issued to remove the row level protection if that is appropriate.

ACTIVATE COLUMN ACCESS CONTROL is ignored if column level access control is already defined as activated for the table.

If the table is a system-period temporal table, the database manager automatically activates row access control on the history table and creates a default row permission for the history table.

DEACTIVATE ROW ACCESS CONTROL

Deactivates row level access control on the table. When the table is referenced in a data manipulation statement, any existing enabled row permissions defined on the table are not applied by the database manager to control the set of rows in the table that are accessible.

DEACTIVATE ROW ACCESS CONTROL is ignored if row access control is not activated for the table.

DEACTIVATE COLUMN ACCESS CONTROL

Deactivates column level access control on the table. When the table is referenced in a data manipulation statement, any existing enabled column masks defined on the table are not applied by the database manager to control the values returned for the columns referenced in the final result table of the queries.

DEACTIVATE COLUMN ACCESS CONTROL is ignored if column access control is not activated for the table.

RENAME COLUMN *source-column-name* **TO** *target-column-name*
Renames the column that is specified in *source-column-name* to the name that is specified in *target-column-name*. If the **auto_reval** database configuration parameter is set to DISABLED, the RENAME COLUMN option of the ALTER TABLE statement behaves like it is under the control of revalidation immediate semantics.

The table must not be defined as a history table (SQLSTATE 42986). If the table is a system-period temporal table, the column is also renamed in any associated history table.

RENAME COLUMN must not rename a column that is referenced in the definition of a row permission or a column mask. Also, It must not rename a column for which a column mask is defined (SQLSTATE 42917). If you rename a column that belongs to a table on which a mask or a permission is defined, or to a table that is referenced in the definition of a mask or a permission, that mask or permission is invalidated. Access to a table that has column access control activated and an invalid mask defined on it is blocked until the invalid mask is either disabled, dropped, or recreated (SQLSTATE 560D0). Access to a table that has row access control activated and an invalid row permission defined on it is blocked until the invalid permission is either disabled, dropped, or recreated (SQLSTATE 560D0).

source-column-name
Specifies the name of the column that is to be renamed. The *source-column-name* must identify an existing column of the table (SQLSTATE 42703). The name must not be qualified. The name must not identify a column that is otherwise being added, altered, or dropped in the same ALTER TABLE statement (SQLSTATE 42711).

target-column-name
The new name for the column. The name must not be qualified. Existing column names or period names in the table must not be used (SQLSTATE 42711).

DROP PRIMARY KEY
Drops the definition of the primary key and all referential constraints dependent on this primary key. The table must have a primary key (SQLSTATE 42888).

DROP FOREIGN KEY *constraint-name*
Drops the referential constraint *constraint-name*. The *constraint-name* must identify a referential constraint (SQLSTATE 42704). For information about implications of dropping a referential constraint see Notes.

DROP UNIQUE *constraint-name*
Drops the definition of the unique constraint *constraint-name* and all referential constraints dependent on this unique constraint. The *constraint-name* must identify an existing UNIQUE constraint (SQLSTATE 42704). For information on implications of dropping a unique constraint, see Notes.

DROP CHECK *constraint-name*
Drops the check constraint *constraint-name*. The *constraint-name* must identify an existing check constraint defined on the table (SQLSTATE 42704).

DROP CONSTRAINT *constraint-name*
Drops the constraint *constraint-name*. The *constraint-name* must identify an existing check constraint, referential constraint, primary key, or unique constraint defined on the table (SQLSTATE 42704). For information about implications of dropping a constraint, see Notes.

DROP COLUMN
Drops the identified column from the table. The table must not be a typed table (SQLSTATE 428DH). The table cannot have data capture enabled (SQLSTATE 42997). If a column is dropped, the table must be reorganized before an update, insert, or delete operation or an index scan can be performed on the table (SQLSTATE 57016). An XML column can only be dropped only if all of the other XML columns in the table are dropped at the same time.

DROP COLUMN must not drop a column that is referenced in the definition of a row permission or a column mask (SQLSTATE 42917). However, a column for

which a column mask is defined can be dropped. When the column is dropped, any column mask defined on that column is also dropped.

column-name
Identifies the column that is to be dropped. The column name must not be qualified. The name must identify a column of the specified table (SQLSTATE 42703). The name must not identify the only column of the table (SQLSTATE 42814), or a column referenced in the definition of a period (SQLSTATE 42817). The name must not identify the last column of the table that is not hidden (SQLSTATE 428GU). The name must not identify a column in a table that is defined as a system-period temporal table or history table (SQLSTATE 428FR). The name must not identify a column that is part of the distribution key, table-partitioning key, or organizing dimensions (SQLSTATE 42997).

CASCADE
Specifies the following actions, based on the object:

- Any views that are dependent on the column being dropped are marked inoperative
- Any indexes, triggers, SQL functions, constraints, or global variables that are dependent on the column being dropped are also dropped
- Any decomposition-enabled XSROBJECTs that are dependent on the table containing the column are made inoperative for decomposition.

A trigger is dependent on the column if it is referenced in the UPDATE OF column list, or anywhere in the triggered action. A decomposition-enabled XSROBJECT is dependent on a table if it contains a mapping of an XML element or attribute to the table. If an SQL function or global variable is dependent on another database object, it might not be possible to drop the function or global variable by means of the CASCADE option. CASCADE is the default.

RESTRICT
Specifies that the column cannot be dropped if any views, indexes, triggers, constraints, or global variables are dependent on the column, or if any decomposition-enabled XSROBJECT is dependent on the table that contains the column (SQLSTATE 42893). A trigger is dependent on the column if it is referenced in the UPDATE OF column list, or anywhere in the triggered action. A decomposition-enabled XSROBJECT is dependent on a table if it contains a mapping of an XML element or attribute to the table. The first dependent object that is detected is identified in the administration log.

Table 15. Cascaded Effects of Dropping a Column

Operation	RESTRICT Effect	CASCADE Effect
Dropping a column that is referenced by a view or a trigger	Dropping the column is not allowed.	The object and all objects that are dependent on that object are dropped.
Dropping a column that is referenced in the key of an index	If all columns that are referenced in the index are dropped in the same ALTER TABLE statement, dropping the index is allowed. Otherwise, dropping the column is not allowed.	The index is dropped.

Table 15. Cascaded Effects of Dropping a Column (continued)

Operation	RESTRICT Effect	CASCADE Effect
Dropping a column that is referenced in a unique constraint	If all columns that are referenced in the unique constraint are dropped in the same ALTER TABLE statement, and the unique constraint is not referenced by a referential constraint, the columns and the constraint are dropped. (The index that is used to satisfy the constraint is also dropped.) Otherwise, dropping the column is not allowed.	The unique constraint and any referential constraints that reference that unique constraint are dropped. (Any indexes that are used by those constraints are also dropped).
Dropping a column that is referenced in a referential constraint	If all columns that are referenced in the referential constraint are dropped in the same ALTER TABLE statement, the columns and the constraint are dropped. Otherwise, dropping the column is not allowed.	The referential constraint is dropped.
Dropping a column that is referenced by a system-generated column that is not being dropped.	Dropping the column is not allowed.	Dropping the column is not allowed.
Dropping a column that is referenced in a check constraint	Dropping the column is not allowed.	The check constraint is dropped.
Dropping a column that is referenced in a decomposition-enabled XSROBJECT	Dropping the column is not allowed.	The XSROBJECT is marked inoperative for decomposition. Re-enabling the XSROBJECT might require readjustment of its mappings; following this, issue an ALTER XSROBJECT ENABLE DECOMPOSITION statement against the XSROBJECT.
Dropping a column that is referenced in the default expression of a global variable	Dropping the column is not allowed.	The global variable is dropped, unless the dropping of the global variable is disallowed because there are other objects, which do not allow the cascade, that depend on the global variable.

DROP RESTRICT ON DROP
Removes the restriction, if there is one, on dropping the table and the table space that contains the table.

DROP DISTRIBUTION
Drops the distribution definition for the table. The table must have a distribution definition (SQLSTATE 428FT). The table space for the table must be defined on a single partition database partition group.

DROP MATERIALIZED QUERY

Changes a materialized query table so that it is no longer considered to be a materialized query table. The table specified by *table-name* must be defined as a materialized query table that is not replicated (SQLSTATE 428EW). The definition of the columns of *table-name* is not changed, but the table can no longer be used for query optimization, and the REFRESH TABLE statement can no longer be used.

If row level access control or column level access control is in effect for the table, this control remains after the table is no longer a materialized query table.

ADD PERIOD period-definition

Adds a period definition to the table.

SYSTEM_TIME (*begin-column-name***, *end-column-name***)**

Defines a system period with the name SYSTEM_TIME. There must not be a column in the table with the name SYSTEM_TIME (SQLSTATE 42711). A table can have only one SYSTEM_TIME period (SQLSTATE 42711). *begin-column-name* must be defined as ROW BEGIN and *end-column-name* must be defined as ROW END (SQLSTATE 428HN).

BUSINESS_TIME (*begin-column-name***, *end-column-name***)**

Defines an application period with the name BUSINESS_TIME. There must not be a column in the table with the name BUSINESS_TIME (SQLSTATE 42711). A table can have only one BUSINESS_TIME period (SQLSTATE 42711). *begin-column-name* and *end-column-name* must both be defined as DATE or TIMESTAMP(p) where p is from 0 to 12 (SQLSTATE 42842), and the columns must be defined as NOT NULL (SQLSTATE 42831). *begin-column-name* and *end-column-name* must not identify a column that is defined with a GENERATED clause (SQLSTATE 428HZ). Business time period columns cannot be added to a table that is in set integrity pending state.

An implicit check constraint is generated to ensure that the value of *end-column-name* is greater than the value of *begin-column-name*. The name of the implicitly created check constraint is DB2_GENERATED_CHECK_CONSTRAINT_FOR_BUSINESS_TIME and must not be the name of an existing check constraint (SQLSTATE 42710).

DROP PERIOD *period-name*

Drops the identified period from the table. The name must not identify a period that was already added or altered in this ALTER TABLE statement (SQLSTATE 42711). Any implicitly generated check constraints for the period (created when the period was defined) and any indexes that reference the period are also dropped.

period-name

Identifies the period. Valid period names are BUSINESS_TIME or SYSTEM_TIME. The period must exist in the table (SQLSTATE 4274M).

When a BUSINESS_TIME period is dropped, all packages with the application-period temporal table dependency type on that table are invalidated. Other dependent objects like views and triggers that record a dependency on the table are also marked as invalid.

SYSTEM_TIME period cannot be dropped if the table is a system-period temporal table (SQLSTATE 428HZ).

DATA CAPTURE
Indicates whether extra information for data replication is to be written to the log.

If the table is a typed table, then this option is not supported (SQLSTATE 428DH for root tables or 428DR for other subtables).

NONE
Indicates that no extra information will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the schema name (implicit or explicit) of the table is longer than 18 bytes, this option is not supported (SQLSTATE 42997).

INCLUDE LONGVAR COLUMNS
Allows data replication utilities to capture changes made to LONG VARCHAR or LONG VARGRAPHIC columns. The clause may be specified for tables that do not have any LONG VARCHAR or LONG VARGRAPHIC columns since it is possible to ALTER the table to include such columns.

ACTIVATE NOT LOGGED INITIALLY
Activates the NOT LOGGED INITIALLY attribute of the table for this current unit of work.

Any changes made to the table by an INSERT, DELETE, UPDATE, CREATE INDEX, DROP INDEX, or ALTER TABLE in the same unit of work after the table is altered by this statement are not logged. Any changes made to the system catalog by the ALTER statement in which the NOT LOGGED INITIALLY attribute is activated are logged. Any subsequent changes made in the same unit of work to the system catalog information are logged.

At the completion of the current unit of work, the NOT LOGGED INITIALLY attribute is deactivated and all operations that are done on the table in subsequent units of work are logged.

If using this feature to avoid locks on the catalog tables while inserting data, it is important that only this clause be specified on the ALTER TABLE statement. Use of any other clause in the ALTER TABLE statement will result in catalog locks. If no other clauses are specified for the ALTER TABLE statement, then only a SHARE lock will be acquired on the system catalog tables. This can greatly reduce the possibility of concurrency conflicts for the duration of time between when this statement is executed and when the unit of work in which it was executed is ended.

If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

If the table is a system-period temporal table or a history table, this option is not supported

For more information about the NOT LOGGED INITIALLY attribute, see the description of this attribute in “CREATE TABLE”.

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back

(SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY attribute was activated is marked inaccessible after the rollback has occurred and can only be dropped. Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

WITH EMPTY TABLE
Causes all data currently in table to be removed. Once the data has been removed, it cannot be recovered except through use of the RESTORE facility. If the unit of work in which this alter statement was issued is rolled back, the table data will not be returned to its original state.

When this action is requested, no DELETE triggers defined on the affected table are fired. The index data is also deleted for all indexes that exist on the table.

A partitioned table with attached data partitions or logically detached partitions cannot be emptied (SQLSTATE 42928).

PCTFREE *integer*
Specifies the percentage of each page that is to be left as free space during a load or a table reorganization operation. The first row on each page is added without restriction. When additional rows are added to a page, at least *integer* percent of the page is left as free space. The PCTFREE value is considered only by the load and table reorg utilities. The value of *integer* can range from 0 to 99. A PCTFREE value of -1 in the system catalog (SYSCAT.TABLES) is interpreted as the default value. The default PCTFREE value for a table page is 0. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

LOCKSIZE
Indicates the size (granularity) of locks used when the table is accessed. Use of this option in the table definition will not prevent normal lock escalation from occurring. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ROW
Indicates the use of row locks. This is the default lock size when a table is created.

BLOCKINSERT
Indicates the use of block locks during insert operations. This means that the appropriate exclusive lock is acquired on the block before insertion, and row locking is not done on the inserted row. This option is useful when separate transactions are inserting into separate cells in the table. Transactions inserting into the same cells can still do so concurrently, but will insert into distinct blocks, and this can impact the size of the cell if more blocks are needed. This option is only valid for MDC tables (SQLSTATE 42613).

TABLE
Indicates the use of table locks. This means that the appropriate share or exclusive lock is acquired on the table, and that intent locks (except intent none) are not used. For partitioned tables, this lock strategy is applied to both the table lock and the data partition locks for any data partitions that are accessed. Use of this value can improve the performance of queries by limiting the number of locks that need to be acquired. However, concurrency is also reduced, because all locks are held over the complete table.

APPEND
Indicates whether data is appended to the end of the table data or placed where free space is available in data pages. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ON Indicates that table data will be appended and information about free space on pages will not be kept. The table must not have a clustered index (SQLSTATE 428CA).

OFF
Indicates that table data will be placed where there is available space. This is the default when a table is created.

The table should be reorganized after setting APPEND OFF since the information about available free space is not accurate and may result in poor performance during insert.

VOLATILE CARDINALITY or NOT VOLATILE CARDINALITY
Indicates to the optimizer whether or not the cardinality of table *table-name* can vary significantly at run time. Volatility applies to the number of rows in the table, not to the table itself. CARDINALITY is an optional keyword. The default is NOT VOLATILE.

VOLATILE
Specifies that the cardinality of table *table-name* can vary significantly at run time, from empty to large. To access the table, the optimizer will use an index scan (rather than a table scan, regardless of the statistics) if that index is index-only (all referenced columns are in the index), or that index is able to apply a predicate in the index scan. The list prefetch access method will not be used to access the table. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

NOT VOLATILE
Specifies that the cardinality of *table-name* is not volatile. Access plans to this table will continue to be based on existing statistics and on the current optimization level.

COMPRESS
Specifies whether or not data compression applies to the rows of the table.

YES
Specifies that row and XML compression are enabled. Insert and update operations on the table will be subject to compression. Index compression will be enabled for new indexes unless explicitly disabled in the CREATE INDEX statement. Existing indexes can be compressed by using the ALTER INDEX statement.

After a table has been altered to enable row compression, all rows in the table can be compressed immediately by performing one of the following actions:

- REORG command
- Online table move
- Data unload and reload

ADAPTIVE
Enables adaptive compression for the table. Data rows are subject to compression with both table-level and page-level compression dictionaries. XML documents in the XML storage object are subject to

compression with a table-level XML compression dictionary. Page-level compression dictionaries are created automatically as rows are inserted or updated. Table-level compression dictionaries are created for both row and XML data automatically after sufficient data is added, unless they already exist.

STATIC
Enables classic row compression for the table. Data rows are subject to compression with a table-level compression dictionary, and XML documents in the XML storage object are subject to compression using a table-level XML compression dictionary. If no table-level compression dictionaries exists for either row or XML data, they will be created automatically after sufficient data is added.

If neither of the preceding two options are specified along with the COMPRESS YES clause, ADAPTIVE is used implicitly.

NO Specifies that data row and XML compression are disabled. Inserted and updated data rows and XML documents in the table will no longer be subject to compression. Any rows and XML documents in the table that are already in compressed format remain in compressed format until they are converted to non-compressed format when they are updated. An offline reorganization of the table decompresses any rows that are remain compressed. If table-level or page-level compression dictionaries exist, they are discarded during table reorganization or truncation (such as, for example, a LOAD REPLACE operation). Index compression is disabled for new indexes created on that table unless explicitly enabled in the CREATE INDEX statement. Index compression for existing indexes can be explicitly disabled by using the ALTER INDEX statement.

VALUE COMPRESSION
This determines the row format that is to be used. Each data type has a different byte count depending on the row format that is used. For more information, see “Byte Counts” in “CREATE TABLE”. An update operation causes an existing row to be changed to the new row format. Offline table reorganization is recommended to improve the performance of update operations on existing rows. This can also result in the table taking up less space. If the row size, calculated using the appropriate column in the table named “Byte Counts of Columns by Data Type” (see “CREATE TABLE”), would no longer fit within the row size limit, as indicated in the table named “Limits for Number of Columns and Row Size In Each Table Space Page Size”, an error is returned (SQLSTATE 54010). If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ACTIVATE
The NULL value is stored using three bytes. This is the same or less space than when VALUE COMPRESSION is not active for columns of all data types, with the exception of CHAR(1). Whether or not a column is defined as nullable has no affect on the row size calculation. The zero-length data values for columns whose data type is VARCHAR, VARGRAPHIC, CLOB, DBCLOB, or BLOB are to be stored using two bytes only, which is less than the storage required when VALUE COMPRESSION is not active. When a column is defined using the COMPRESS SYSTEM DEFAULT option, this also allows the system default value for the column to be stored using three bytes of total storage. The row format that is used to

support this determines the byte counts for each data type, and tends to cause data fragmentation when updating to or from NULL, a zero-length value, or the system default value.

DEACTIVATE
The null value is stored with space set aside for possible future updates. This space is not set aside for varying-length columns. It also does not support efficient storage of system default values for a column. If columns already exist with the COMPRESS SYSTEM DEFAULT attribute, a warning is returned (SQLSTATE 01648).

LOG INDEX BUILD
Specifies the level of logging that is to be performed during create, re-create, or reorganize index operations on this table.

NULL
Specifies that the value of the **logindexbuild** database configuration parameter will be used to determine whether or not index build operations are to be completely logged. This is the default when the table is created.

OFF
Specifies that any index build operations on this table will be logged minimally. This value overrides the setting of the **logindexbuild** database configuration parameter.

ON Specifies that any index build operations on this table will be logged completely. This value overrides the setting of the **logindexbuild** database configuration parameter.

ADD PARTITION *add-partition*
Adds one or more data partitions to a partitioned table. If the specified table is not a partitioned table, an error is returned (SQLSTATE 428FT). The number of data partitions must not exceed 32 767.

partition-name
Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec
Specifies the range of values for the new data partition. This range must not overlap that of an existing data partition (SQLSTATE 56016). For a description of the starting-clause and the ending-clause, see "CREATE TABLE".

If the starting-clause is omitted, the new data partition is assumed to be at the end of the table. If the ending-clause is omitted, the new data partition is assumed to be at the start of the table.

IN *tablespace-name*
Specifies the table space where the data partition is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838). This can be a table space that is already being used for another data partition of the same table, or a table space that is currently not being used by this table, but it must be a table space on which the authorization ID of the statement holds the USE privilege (SQLSTATE 42727). If this clause is not specified, the table space of the first visible or attached data partition of the table is used.

INDEX IN *tablespace-name*
Specifies the table space where partitioned indexes on the data partition are stored. If the INDEX IN clause is not specified, partitioned indexes on the data partition are stored in the same table space as the data partition.

The table space used by the new index partition, whether default or specified by the INDEX IN clause, must match the type (SMS or DMS), page size, and extent size of the table spaces used by all other index partitions (SQLSTATE 42838).

LONG IN *tablespace-name*
Specifies the table space where the data partition containing long column data is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces and data partitions of the partitioned table (SQLSTATE 42838); it must be a table space on which the authorization ID of the statement holds the USE privilege. The page size and extent size for the named table space can be different from the page size and extent size of the other data partitions of the partitioned table.

For rules governing the use of the LONG IN clause with partitioned tables, see "Large object behavior in partitioned tables".

ATTACH PARTITION *attach-partition*
Attaches another table as a new data partition. The data object of the table being attached becomes a new partition of the table being attached to. There is no data movement involved. The table is placed in set integrity pending state, and referential integrity checking is deferred until execution of a SET INTEGRITY statement. The ALTER TABLE ATTACH operation does not allow the use of the IN or LONG IN clause. The placement of LOBs for that data partition is determined at the time the source table is created. For rules governing the use of the LONG IN clause with partitioned tables, see "Large object behavior in partitioned tables".

If the table being attached has either row level access control or column level access control activated then the table to attach to must have the same controls activated. No row permissions or column masks are automatically carried over from the table being attached to the target table. The column masks and row permissions do not necessarily need to be exactly the same on both tables, although this would be best from a security perspective. But if the table being attached has row level access control activated then the table to attach to must also have row level access control activated (SQLSTATE 428GE). Similarly, if the table being attached has column level access control activated and at least one column mask object enabled then the table to attach to must also have column level access control activated and a column mask object enabled for the corresponding columns (SQLSTATE 428GE).

partition-name
Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec
Specifies the range of values for the new data partition. This range must not overlap that of an existing data partition (SQLSTATE 56016). For a description of the starting-clause and the ending-clause, see "CREATE TABLE".

If the starting-clause is omitted, the new data partition is assumed to be at the end of the table. If the ending-clause is omitted, the new data partition is assumed to be at the start of the table.

FROM *table-name1*
Specifies the table that is to be used as the source of data for the new partition. The table definition of *table-name1* cannot have multiple data partitions, and it must match the altered table in the following ways (SQLSTATE 428GE):

- The number of columns must be the same.
- The data types of the columns in the same ordinal position in the table must be the same.
- The nullability characteristic of the columns in the same ordinal position in the table must be the same.
- If the target table has a row change timestamp column, the corresponding column of the source table must be a row change timestamp column.
- If the data is also distributed, it must be distributed over the same database partition group using the same distribution method.
- If the data in either table is organized, the organization must match.
- For structured, XML, or LOB data type, the value for INLINE LENGTH must be the same.
- If the target table has a BUSINESS_TIME period defined, the source table must have a BUSINESS_TIME period defined on the corresponding columns.

After the data from *table-name1* is successfully attached, an operation equivalent to DROP TABLE *table-name1* is performed to remove this table, which no longer has data, from the database.

BUILD MISSING INDEXES
Specifies that if the source table does not have indexes that correspond to the partitioned indexes on the target table, a SET INTEGRITY operation builds partitioned indexes on the new data partition to correspond to the partitioned indexes on the existing data partitions. Indexes on the source table that do not match the partitioned indexes on the target table are dropped during attach processing.

REQUIRE MATCHING INDEXES
Specifies that the source table must have indexes to match the partitioned indexes on the target table; otherwise, an error is returned (SQLSTATE 428GE) and information is written to the administration log about the indexes that do not match.

If the REQUIRE MATCHING INDEXES clause is not specified and the indexes on the source table do not match all the partitioned indexes on the target table, the following behavior occurs:

1. For indexes on the target table that do not have a match on the source table and are either unique indexes or XML indexes that are defined with REJECT INVALID VALUES, the ATTACH operation fails (SQLSTATE 428GE).
2. For all other indexes on the target table that do not have a match on the source table, the index object on the source table is marked invalid during the attach operation. If the source table does not have any indexes, an empty index object is created and marked as invalid. The ATTACH operation will succeed, but the index object on the new data

partition is marked as invalid. Typically, SET INTEGRITY is the next operation to run against the data partition. SET INTEGRITY will force a rebuild, if required, of the index object on data partitions that were recently attached. The index rebuild can increase the time required to bring the new data online.

3. Information is written to the administration log about the indexes that do not match.

DETACH PARTITION *partition-name* **INTO** *table-name1*

Detaches the data partition *partition-name* from the altered table, and uses the data partition to create a new table named *table-name1*. The data partition is detached from the altered table and is used to create the new table without any data movement. The specified data partition cannot be the last remaining partition of the table being altered (SQLSTATE 428G2). The table being altered to detach a partition must not be a system-period temporal table (SQLSTATE 428HZ).

When a partition is detached from a table for which either row level access control or column level access control is defined, the new table that is created for the detached data will automatically have row level access control (though not column level access control) activated to protect the detached data. Direct access to this new table will return no rows until appropriate row permissions are defined for the table or row level access control is deactivated for this table.

ADD SECURITY POLICY *policy-name*

Adds a security policy to the table. The security policy must exist at the current server (SQLSTATE 42704). The table must not already have a security policy (SQLSTATE 55065), and must not be a typed table (SQLSTATE 428DH), materialized query table (MQT), or staging table (SQLSTATE 428FG).

DROP SECURITY POLICY

Removes the security policy and all LBAC protection from the table. The table specified by *table-name* must be protected by a security policy (SQLSTATE 428GT). If the table has a column with data type DB2SECURITYLABEL, the data type is changed to VARCHAR (128) FOR BIT DATA. If the table has one or more protected columns, those columns become unprotected.

ADD VERSIONING USE HISTORY TABLE *history-table-name*

Specifies that the table is a system-period temporal table. The table must not already be defined as a system-period temporal table or a history table (SQLSTATE 428HM). A SYSTEM_TIME period and a transaction-start-ID column must be defined in the table (SQLSTATE 428HM). The table must not be a materialized query table (SQLSTATE 428HM).

Historical versions of the rows in the table are retained by the database manager. The database manager records extra information that indicates when a row was inserted into the table, and when it was updated or deleted. When a row in a system-period temporal table is updated, a previous version of the row is kept. When data in a system-period temporal table is deleted, the old version of the row is inserted as a historical record. An associated history table is used to store the historical rows of the table.

References to the table can include a time period search condition to indicate which system versions of the data are to be returned.*history-table-name* identifies a history table where historical rows of the system-period temporal table are kept. *history-table-name* must identify a table that exists at the current server (SQLSTATE 42704), and is not a catalog table (SQLSTATE 42832), an

existing system-period temporal table, an existing history table, a declared global temporary table, a created global temporary table, a materialized query table, or a view, (SQLSTATE 428HX).

The identified history table must not contain an identity column, row change timestamp column, row-begin column, row-end column, transaction start-ID column, generated expression column, or include a period (SQLSTATE 428HX).

The system-period temporal table and the identified history table must have the same number and order of columns (SQLSTATE 428HX). The following attributes for the corresponding columns of the two tables must be the same (SQLSTATE 428HX):

- Column name
- Column data type
- Column length (including inline LOB lengths), precision, and scale
- Column FOR BIT attribute for character string columns
- Column null attribute
- Column hidden attribute

If row access control or column access control is activated for the system-period temporal table and row access control is not activated on the history table, the database manager automatically activates row access control on the history table and creates a default row permission for the history table.

DROP VERSIONING

Specifies that the table is no longer a system-period temporal table. The table must be a system-period temporal table (SQLSTATE 428HZ). Historical data is no longer recorded and maintained for the table. The definition of the columns and data of the table are not changed, but the table is no longer treated as a system-period temporal table. The SYSTEM_TIME period is retained. Subsequent queries that reference the table must not specify a SYSTEM_TIME period specification for the table. The relationship between the system-period temporal table and the associated history table is removed. The history table is not dropped and the contents of the history table are not affected.

When a table is altered with DROP VERSIONING, all packages with the system-period temporal table dependency type on that table are invalidated. Other dependent objects like views and triggers that record a dependency on the table are also marked as invalid.

Rules

- Any unique or primary key constraint defined on the table must be a superset of the distribution key, if there is one (SQLSTATE 42997).
- Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).
- A column can only be referenced in one ADD, ALTER, or DROP COLUMN clause in a single ALTER TABLE statement (SQLSTATE 42711).
- A column length, data type, or hidden attribute cannot be altered, nor can the column be dropped, if the table has any materialized query tables that are dependent on the table (SQLSTATE 42997).
- VARCHAR and VARGRAPHIC columns that have been altered to be greater than 4000 and 2000, respectively, must not be used as input parameters in functions in the SYSFUN schema (SQLSTATE 22001).
- A column length cannot be altered if the table has any views enabled for query optimization that are dependent on the table (SQLSTATE 42997).

- The table must be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option (SQLSTATE 55019), before:
  - Adding a column with a generation expression
  - Altering the generated expression of a column
  - Changing a column to have a generated expression
- An existing column cannot be altered to become of type DB2SECURITYLABEL (SQLSTATE 42837).
- Defining a column of type DB2SECURITYLABEL fails if the table does not have a security policy associated with it (SQLSTATE 55064).
- A column of type DB2SECURITYLABEL cannot be altered or dropped (SQLSTATE 42817).
- An ALTER TABLE operation to mark a table as protected fails if there exists an MQT that depends on that table (SQLSTATE 55067).
- Attaching a partition to a protected partitioned table fails if the source table and the target table are not protected using the same security policy, do not have the same row security label column, and do not have the same set of protected columns (SQLSTATE 428GE).
- If a generated column is referenced in a table-partitioning key, the generated column expression cannot be altered (SQLSTATE 42837).
- The *isolation-clause* cannot be specified in the *fullselect* of the *materialized-query-definition* (SQLSTATE 42601).
- Adding or attaching a data partition to a partitioned table fails with SQL0612N after detaching the same partition name, if asynchronous index cleanup has not finished to delete index entries for the partition (SQLSTATE 42711).

Notes

- A *REORG-recommended operation* has occurred when changes resulting from an ALTER TABLE statement affect the row format of the data. When this occurs, most subsequent operations on the table are restricted until a table reorganization operation completes successfully. Up to three ALTER TABLE statements of this type can execute against a table before reorganization must be done (SQLSTATE 57016). Multiple alterations that would constitute a REORG-recommended operation can be made as part of a single ALTER TABLE statement (one per column); this is considered to be a single REORG-recommended operation. For example, dropping two columns in a single ALTER TABLE statement is not considered to be two REORG-recommended operations. Dropping two columns in two separate ALTER TABLE statements, however, would be regarded as two statements that contain REORG-recommended operations.
- The following table operations are allowed after a successful REORG-recommended operation has occurred:
  - ALTER TABLE, where no row data validation is required. However, the following operations are not allowed (SQLSTATE 57007):
    - ADD CHECK CONSTRAINT
    - ADD REFERENTIAL CONSTRAINT
    - ADD UNIQUE CONSTRAINT
    - ALTER COLUMN SET NOT NULL
  - DROP TABLE
  - RENAME TABLE
  - REORG TABLE

  - TRUNCATE TABLE
  - Table scan access of table data
- Altering a table to make it a materialized query table will put the table in set integrity pending state. If the table is defined as REFRESH IMMEDIATE, the table must be taken out of set integrity pending state before INSERT, DELETE, or UPDATE commands can be invoked on the table referenced by the fullselect. The table can be taken out of set integrity pending state by using REFRESH TABLE or SET INTEGRITY, with the IMMEDIATE CHECKED option, to completely refresh the data in the table based on the fullselect. If the data in the table accurately reflects the result of the fullselect, the IMMEDIATE UNCHECKED option of SET INTEGRITY can be used to take the table out of set integrity pending state.
- Altering a table to change it to a REFRESH IMMEDIATE materialized query table will cause any packages with INSERT, DELETE, or UPDATE usage on the table referenced by the fullselect to be invalidated.
- Altering a table to change from a materialized query table to a regular table will cause any packages dependent on the table to be invalidated.
- Altering a table to change from a MAINTAINED BY FEDERATED_TOOL materialized query table to a regular table will not cause any change in the subscription setup of the replication tool. Because a subsequent change to a MAINTAINED BY SYSTEM materialized query table will cause the replication tool to fail, you must change the subscription setting when changing a MAINTAINED BY FEDERATED_TOOL materialized query table.
- If a deferred materialized query table is associated with a staging table, the staging table will be dropped if the materialized query table is altered to a regular table.
- ADD column clauses are processed before all other clauses. Other clauses are processed in the order that they are specified.
- Any columns added through an alter table operation will not automatically be added to any existing view of the table.
- Adding or attaching a data partition to a partitioned table, or detaching a data partition from a partitioned table, causes any packages that are dependent on that table to be invalidated.
- For DB2 Version 9.7 Fix Pack 1 and later releases, after detaching a data partition from a data partitioned table, the STATUS of the detached partition in the SYSCAT.DATAPARTITIONS catalog can be 'L' when the partition is logically detached and the detach operation has not completed. If the STATUS of the detached partition is 'L', the following operations cannot be performed on the source table (SQLSTATE 55057):
  - Adding a unique or primary key constraint that attempts to create a nonpartitioned index
  - Adding, dropping, or renaming a column
  - Activating value compression or compression
  - Deactivating value compression or compression
- To drop the partitioning for a table, the table must be dropped and then recreated.
- To drop the organization for a table, the table must be dropped and then recreated.
- When an index is automatically created for a unique or primary key constraint, the database manager will try to use the specified constraint name as the index name with a schema name that matches the schema name of the table. If this matches an existing index name or no name for the constraint was specified, the

index is created in the SYSIBM schema with a system-generated name formed of "SQL" followed by a sequence of 15 numeric characters generated by a timestamp based function.

- When a nonpartitioned index is created on a partitioned table with attached data partitions, the index will not include the data in the attached data partitions. Use the SET INTEGRITY statement to maintain all indexes for all attached data partitions.
- When creating a partitioned index in the presence of attached data partitions (STATUS of 'A' in SYSCAT.DATAPARTITIONS), an index partition for each attached data partition will also be created. If the partitioned index is being created as unique, or is an XML index being created with REJECT INVALID VALUES, then the index creation can fail if an attached data partition contains any violations (duplicates for a unique index, or invalid values for the XML index).
- If a table has a nonpartitioned index, you cannot access a new data partition in that table within the same transaction as the add or attach operation that created the partition, if the transaction does not have the table locked in exclusive mode (SQLSTATE 57007).
- Any table that may be involved in a DELETE operation on table T is said to be *delete-connected* to T. Thus, a table is delete-connected to T if it is a dependent of T or it is a dependent of a table in which deletes from T cascade.
- A package has an insert (update/delete) usage on table T if records are inserted into (updated in/deleted from) T either directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements. Similarly, a package has an update usage on a column if the column is modified directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements.
- In a federated system, a remote base table that was created using transparent DDL can be altered. However, transparent DDL does impose some limitations on the modifications that can be made:
  - A remote base table can only be altered by adding new columns or specifying a primary key.
  - Specific clauses supported by transparent DDL include:
    - ADD COLUMN *column-definition*
    - NOT NULL and PRIMARY KEY in the *column-options* clause
    - ADD *unique-constraint* (PRIMARY KEY only)
  - You cannot specify a comment on an existing column in a remote base table.
  - An existing primary key in a remote base table cannot be altered or dropped.
  - Altering a remote base table invalidates any packages that are dependent on the nickname associated with that remote base table.
  - The remote data source must support the changes being requested through the ALTER TABLE statement. Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.
  - An attempt to alter a remote base table that was not created using transparent DDL returns an error.
- Any changes, whether implicit or explicit, to primary key, unique keys, or foreign keys might have the following effects on packages, indexes, and other foreign keys.
  - If a primary key or unique key is added:

    - There is no effect on packages, foreign keys, or existing unique keys. (If the primary or unique key uses an existing unique index that was created in a previous version and has not been converted to support deferred uniqueness, the index is converted, and packages with update usage on the associated table are invalidated.)
  - If a primary key or unique key is dropped:
    - The index is dropped if it was automatically created for the constraint. Any packages dependent on the index are invalidated.
    - The index is set back to non-unique if it was converted to unique for the constraint and it is no longer system-required. Any packages dependent on the index are invalidated.
    - The index is set to no longer system required if it was an existing unique index used for the constraint. There is no effect on packages.
    - All dependent foreign keys are dropped. Further action is taken for each dependent foreign key, as specified in the next item.
  - If a foreign key is added, dropped, or altered from NOT ENFORCED to ENFORCED (or ENFORCED to NOT ENFORCED):
    - All packages with an insert usage on the object table are invalidated.
    - All packages with an update usage on at least one column in the foreign key are invalidated.
    - All packages with a delete usage on the parent table are invalidated.
    - All packages with an update usage on at least one column in the parent key are invalidated.
  - If a foreign key or a functional dependency is altered from ENABLE QUERY OPTIMIZATION to DISABLE QUERY OPTIMIZATION:
    - All packages with dependencies on the constraint for optimization purposes are invalidated.
- Adding a column to a table will result in invalidation of all packages with insert usage on the altered table. If the added column is the first user-defined structured type column in the table, packages with DELETE usage on the altered table will also be invalidated.
- Adding a check or referential constraint to a table that already exists and that is not in set integrity pending state, or altering the existing check or referential constraint from NOT ENFORCED to ENFORCED on an existing table that is not in set integrity pending state will cause the existing rows in the table to be immediately evaluated against the constraint. If the verification fails, an error is returned (SQLSTATE 23512). If a table is in set integrity pending state, adding a check or referential constraint, or altering a constraint from NOT ENFORCED to ENFORCED will not immediately lead to the enforcement of the constraint. Issue the SET INTEGRITY statement with the IMMEDIATE CHECKED option to begin enforcing the constraint.
- Adding, altering, or dropping a check constraint will result in invalidation of all packages with either an insert usage on the object table, an update usage on at least one of the columns involved in the constraint, or a select usage exploiting the constraint to improve performance.
- Adding a distribution key invalidates all packages with an update usage on at least one of the columns of the distribution key.
- A distribution key that was defined by default as the first column of the primary key is not affected by dropping the primary key and adding a different primary key.

- Dropping a column or changing its data type removes all runstats information from the table being altered. Runstats should be performed on the table after it is again accessible. The statistical profile of the table is preserved if the table does not contain a column that was explicitly dropped.
- Altering a column (to change its length, data type, nullability, or hidden attribute) or dropping a column invalidates all packages that reference (directly or indirectly through a referential constraint or trigger) its table.
- Altering a column (to change its length, data type, nullability, or hidden attribute) regenerates views (except typed views) that are dependent on its table. If a problem occurs while regenerating such a view, an error is returned (SQLSTATE 56098). Any typed views that are dependent on the table are marked inoperative.
- Altering a column (to change its length, data type, or hidden attribute) marks all dependent triggers and SQL functions as invalid; they are implicitly recompiled on next use. If a problem occurs while regenerating such an object, an error is returned (SQLSTATE 56098).
- Altering a column (to change its length, data type, or nullability attribute) might cause errors (SQLSTATE 54010) while processing a trigger or an SQL function when a statement involving the trigger or SQL function is prepared or bound. This can occur if the row size based on the sum of the lengths of the transition variables and transition table columns is too long. If such a trigger or SQL function is dropped, a subsequent attempt to re-create it returns an error (SQLSTATE 54040).
- Starting with DB2 Version 9.7 Fix Pack 1, a WLM activity event monitor created in an earlier version must be dropped and re-created to add new table columns introduced by this fix pack and any subsequent fix packs or releases.
- Altering a structured or XML type column to increase the inline length will invalidate all packages that reference the table, either directly or indirectly through a referential constraint or trigger.
- Altering a structured or XML type column to increase the inline length will regenerate views that are dependent on the table.
- A compression dictionary can be created for the XML storage object of a table only if the XML columns are added to the table in DB2 Version 9.7 or later, or if the table is migrated the using an online table move.
- Changing the LOCKSIZE for a table will result in invalidation of all packages that have a dependency on the altered table.
- Changing VOLATILE or NOT VOLATILE CARDINALITY will result in invalidation of all packages that have a dependency on the altered table.
- ***Replication:*** Exercise caution when increasing the length or changing the data type of a column. The change data table that is associated with an application table might already be at or near the DB2 row size limit. The change data table should be altered before the application table, or the two tables should be altered within the same unit of work, to ensure that the alteration can be completed for both tables. Consideration should be given to copies, which might also be at or near the row size limit, or reside on platforms which lack the ability to increase the length of an existing column.

  If the change data table is not altered before the Capture program processes log records with the altered attributes, the Capture program will likely fail. If a copy containing the altered column is not altered before the subscription maintaining the copy runs, the subscription will likely fail.
- When detaching a partition from a protected table, the target table automatically created by DB2 will be protected in exactly the same way the source table is protected.

- When a table is altered such that it becomes protected with row level granularity, any cached dynamic SQL sections that depend on such a table are invalidated. Similarly, any packages that depend on such a table are also invalidated.
- When a column of a table, T, is altered such that it becomes a protected column, any cached dynamic SQL sections that depend on table T are invalidated. Similarly, any packages that depend on table T are also invalidated.
- When a column of a table, T, is altered such that it becomes a non protected column, any cached dynamic SQL sections that depend on table T are invalidated. Similarly, any packages that depend on table T are also invalidated.
- For existing rows in the table, the value of the security label column defaults to the security label for write access of the session authorization ID at the time the ALTER statement that adds a row security label column is executed.
- ***Add materialized query:*** When a base table is altered to become a materialized query table, the label-based access control security attributes (security policy, column security labels, row security label column) are derived in the same way when creating a new materialized query table. If the base table that is altered already has label-based access control security attributes, these attributes are factored in the derivation process as follows:
  - Column access control: The existing security label for a column is aggregated with the corresponding security label derived from the query defining the materialized query table.
  - Row access control: The row access control attributes are setup exactly in the same way as for a new materialized query table.
- In DB2 Version 9.7 Fix Pack 1 or later releases, new multidimensional clustering (MDC) table block indexes are partitioned. Adding a data partition to a data partitioned multidimensional clustering (MDC) table creates the corresponding empty index partitions for the new partition, including the MDC block indexes. Also, a new index partition entry is added to SYSCAT.SYSINDEXPARTITIONS for each MDC block index, as well as for each partitioned index.
- When attaching a data partition to a partitioned MDC table created with DB2 V9.7 Fix Pack 1 or later releases, the source table specified by *attach-partition* can be a nonpartitioned MDC table or a single-partition partitioned MDC table.
  - **If the source table is nonpartitioned:** MDC block indexes on the source table will be inherited and become the partitioned MDC indexes for the new partition after the ATTACH operation completes.
  - **If the source table is partitioned:** If the source table is a partitioned MDC table created with DB2 V9.7 Fix Pack 1 or later releases, the block indexes are partitioned. The block indexes become the new block indexes on the partition.
  - If the source partitioned MDC table is created at a level lower than DB2 V9.7 Fix Pack 1, the block indexes on the table are nonpartitioned. During the ATTACH operation, the block indexes are dropped and created as partitioned indexes similar to the other partitioned indexes on the source table.

    Issuing the SET INTEGRITY statement on the target table is required to bring the attached partition online.

    If the REQUIRE MATCHING INDEXES clause is specified, and the target table is a partitioned MDC table created in DB2 V9.7 Fix Pack 1 or later releases, the ALTER TABLE ... ATTACH PARTITION statement fails and returns SQL20307N (SQLSTATE 428GE). Removing the REQUIRE MATCHING INDEXES clause allows the attach process to proceed.

  If the target partitioned MDC table was created at a level lower than DB2 V9.7 Fix Pack 1, the block indexes are nonpartitioned. The block indexes on the

source MDC table are dropped during the ATTACH operation. Issuing a SET INTEGRITY statement on the target table is required to bring the attached partition online. New rows from the attached partition are added to existing nonpartitioned block indexes.

- When detaching a data partition from a data partitioned MDC table created at a level lower than DB2 V9.7 Fix Pack 1, the block indexes are nonpartitioned. The following restrictions apply:
  - Access to the newly detached table is not allowed in the same unit of work as the detach operation.
  - Block indexes on the target table, created as part of the detach operation, are rebuilt upon the first access to the table after the detach operation is committed. If the source table had any partitioned indexes before the detach operation then the index object for the target table will be marked invalid to allow for recreation of the block indexes. As a result, access time is increased while the block indexes and all other partitioned indexes are re-created.

  When detaching a partition from a partitioned MDC table created using DB2 V9.7 Fix Pack 1 or later releases, the block indexes are partitioned, and the previous restrictions do not apply. Assuming that no other dependent objects such as dependent MQTs exist, access to the newly detached table is allowed in the same unit of work. All the partitioned indexes, including block indexes, become indexes on the target table without the need to be re-created.
- ***Considerations for implicitly hidden columns:*** A column that is defined as implicitly hidden can be explicitly referenced in an ALTER TABLE statement. For example, an implicitly hidden column can be altered or specified as part of a referential constraint, check constraint, or materialized query table definition.

  Altering a table to make some of its columns implicitly hidden can impact the behavior of data movement utilities that are working with the table. When a table contains implicitly hidden columns, utilities like IMPORT, INGEST, and LOAD require that you specify whether data for the hidden columns is included in the operation. For example, this might mean that a load operation that ran successfully before the table was altered, now fails (SQLCODE SQL2437N). Similarly, EXPORT requires that you specify whether data for the hidden columns is included in the operation.

  Data movement utilities must use the DB2_DMU_DEFAULT registry variable, or the **implicitlyhiddeninclude** or **implicitlyhiddenmissing** file type modifiers when working with tables that contain implicitly hidden columns.
- ***Row access control that is activated explicitly***: The ACTIVATE ROW ACCESS CONTROL clause is used to activate row access control for a table. When this happens, a default row permission is implicitly created and allows no access to any rows of the table, unless permitted by a row permission explicitly created by the security administrator. The default row permission is always enabled.

  When the table is referenced in a data manipulation statement, all enabled row permissions that have been created for the table, including the default row permission, are implicitly applied by the database manager to control which rows in the table are accessible. A row access control search condition is derived by application of the logical `OR` operator to the search condition in each enabled row permission. This derived search condition acts as a filter to the table before any user specified operations, such as predicates, grouping, ordering, and so on, are processed. This derived search condition permits the authorization IDs that are specified in the permission definitions to access certain rows in the table.

  When the ACTIVATE ROW ACCESS CONTROL clause is used, all the packages and dynamically cached statements that reference the table are invalidated.

Row access control remains enforced until the DEACTIVATE ROW ACCESS CONTROL clause is used to stop enforcing it.

- ***Implicit object that is created when row access control is activated for a table***: When the ACTIVATE ROW ACCESS CONTROL clause is used to activate row access control for a table, the database manager implicitly creates a default row permission for the table. The default row permission prevents all access to the table. The implicitly created row permission resides in the same schema of the base table and has a name in the form of SYS_DEFAULT_ROW_PERMISSION__table-name ... up to 128 characters. Notice two underscores after "PERMISSION". If this name is not unique, the last 4 characters are reserved for a unique number *'nnnn'*, where *'nnnn'* is a four alphanumeric character string starting at '0000' and is incremented by 1 value each time until a unique name is found.

  The owner of the default row permission is SYSIBM. The default row permission is always enabled. The default row permission is dropped when row access control is deactivated or when the table is dropped.
- ***Activating column access control:*** The ACTIVATE COLUMN ACCESS CONTROL clause is used to activate column level access control for a table. The access to the table is not restricted but when the table is referenced in a data manipulation statement, all enabled column masks that have been created for the table are applied to mask the column values referenced in the final result table.

  When column masks are used to mask the column values, they determine the values in the final result table. If a column has a column mask and the column (specifically a simple reference to a column name or a column embedded in an expression) appears in the outermost select list, the column mask is applied to the column to produce the values for the final result table. If the column does not appear in the outermost select list but it participates in the final result table, for example, it appears in a materialized table expression or view, the column mask is applied to the column in such a way that the masked value is included in the result table of the materialized table expression or view so that it can be used in the final result table.

  The application of column masks does not interfere with the operations of other clauses within the statement such as the WHERE, GROUP BY, HAVING, SELECT DISTINCT, and ORDER BY. The rows returned in the final result table remain the same, except that the values in the resulting rows may have been masked by the column masks. As such, if the masked column also appears in an ORDER BY *sort-key*, the order is based on the original column values and the masked values in the final result table may not reflect that order. Similarly, the masked values may not reflect the uniqueness enforced by SELECT DISTINCT.

  A column mask is applied in the following contexts:

  – The outermost SELECT clause or clauses of a SELECT or SELECT INTO statement, or if the column does not appear in the outermost select list butit participates in the final result table, the outermost SELECT clause(s) of the corresponding materialized table expression or view where the column appears.

  – The outermost SELECT clause or clauses of a SELECT FROM INSERT, SELECT FROM UPDATE, or SELECT FROM DELETE operation.

  – The outermost SELECT clause or clauses that are used to derive the new values for an INSERT, UPDATE, or MERGE statement, or a SET *transition-variable-name* assignment statement. The same masking applies to a scalar fullselect expression that appears in the outermost SELECT clause or

clauses of the previously mentioned statements, the right side of a SET *host-variable* assignment statement, the VALUES INTO statement, or the VALUES statement.

Column masks are not applied when the masked column appears in the following contexts:

- WHERE clauses.
- GROUP BY clauses.
- HAVING clauses.
- SELECT DISTINCT.
- ORDER BY clauses.

- ***Row and column access control are not enforced when EXPLAIN tables are populated:*** Row and column access control can be enforced for EXPLAIN tables. However, the enabled row permissions and column masks are not applied when the database manager inserts rows into those tables.
- ***Row and column access control are not enforced when event monitor tables are populated:*** Row and column access control can be enforced for event monitor tables. However, the enabled row permissions and column masks are not applied when the database manager inserts rows into those tables.
- ***Row and column access control are not enforced when temporal history tables are populated:*** Row and column access control can be enforced for temporal history tables. However, the enabled row permissions and column masks are not applied when the database manager accesses those tables for operations on the system-period temporal tables.
- ***Stop enforcing row or column access control:*** The DEACTIVATE ROW ACCESS CONTROL clause is used to stop enforcing row access control for a table. The default row permission is dropped. Thereafter, when the table is referenced in a data manipulation statement, explicitly created row permissions are not applied.

  The DEACTIVATE COLUMN ACCESS CONTROL clause is used to stop enforcing column access control for a table. Thereafter, when the table is referenced in a data manipulation statement, the column masks are not applied.

  The explicitly created row permissions or column masks, if any, remain but have no effect.

  All the packages and dynamically cached statements that reference the table are invalidated when row or column access control is deactivated.
- ***Secure triggers for row and column access control:*** Triggers are used for database integrity, and as such, a balance between row and column access control (security) and database integrity is needed. Enabled row permissions and column masks are not applied to the initial values of transition variables and transition tables. Row and column access control enforced for the triggering table is also ignored for any transition variables or transition tables referenced in the trigger body. To ensure there is no security concern for SQL statements in the trigger action to access sensitive data in transition variables and transition tables, the trigger must be created or altered with the SECURED option. If a trigger is not secure, row and column access control cannot be enforced for the triggering table (SQLSTATE 55019).
- ***Secure user-defined functions for row and column access control:*** If a row permission or column mask definition references a user-defined function, the function must be altered with the SECURED option because the sensitive data may be passed as arguments to the function. When a user-defined function is referenced in a data manipulation statement where a table that enforces row or column access control is referenced, and the function arguments reference the columns from such a table, if the function is not secure, this impacts the access

plan selection and may yield poor performance. The database manager considers the SECURED option an assertion that declares the user has established a change control audit procedure for all changes to the user-defined function. It is assumed that such a control audit procedure is in place and that all subsequent ALTER FUNCTION statements or changes to external packages are being reviewed by this audit process.

- ***Database operations where row and column access control is not applicable:*** Row and column access control must not compromise database integrity. Columns involved in primary keys, unique keys, indexes, check constraints, and referential integrity must not be subject to row and column access control. Column masks can be defined for those columns but they are not applied during the process of key building or constraint or RI enforcement.
- ***Defining a system-period temporal table:*** A system-period temporal table definition includes the following aspects:
  - A system period named SYSTEM_TIME which is defined using a row-begin column and a row-end column. See the descriptions of AS ROW BEGIN, AS ROW END, and period-definition.
  - A transaction-start-ID column. See the description of AS TRANSACTION START ID.
  - A system-period data versioning definition specified on a subsequent ALTER TABLE statement using the ADD VERSIONING action which includes the name of the associated history table. See the description of the ADD VERSIONING clause under ALTER TABLE.

  To ensure that the history table cannot be implicitly dropped when a system-period temporal table is dropped, use the WITH RESTRICT ON DROP clause in the definition of the history table.
- ***Defining an application-period temporal table:*** An application-period temporal table definition includes an application period with the name BUSINESS_TIME. The application period is defined using a begin column and an end column with both columns having the same data type that is either DATE or TIMESTAMP(**p**). See the description of period-definition.

  Data change operations on an application-period temporal table may result in an automatic insert of one or two additional rows when a row is updated or deleted. When an update or delete of a row in an application-period temporal table is specified for a portion of the period represented by that row, the row is updated or deleted and one or two rows are automatically inserted to represent the portion of the row that is not changed. New values are generated for each generated column in an application-period temporal table for each row that is automatically inserted as a result of an update or delete operation on the table. If a generated column is defined as part of a unique or primary key, parent key in a referential constraint, or unique index, it is possible that an automatic insert will violate a constraint or index in which case an error is returned.
- ***Considerations for transaction-start-ID columns:*** A transaction-start-ID column contains a null value if the column allows null values, and there is a row-begin column and the value of the row-begin column is unique from values of row-begin columns generated for other transactions. Given that the column may contain null values, it is recommended that one of the following methods be used when retrieving a value from the column:
  - COALESCE (transaction_start_id_col, row_begin_col)
  - CASE WHEN transaction_start_id_col IS NOT NULL THEN transaction_start_id_col ELSE row_begin_col END

- ***Considerations for system-period temporal tables and row and column access control:*** Row and column access control can be defined on both the system-period temporal table and the associated history table.
  - When a system-period temporal table is accessed, any row and column access rules defined on the system-period temporal table are applied to all of the rows returned from the system-period temporal table, regardless of whether the rows are stored in the system-period temporal table or the history table. The row and column access rules defined on the history table are not applied.
  - When the history table is accessed directly, the row and column access rules defined on the history table are applied.

  When a system-period temporal table is defined and row access control or column access control is activated for the system-period temporal table, the database manager automatically activates row access control on the history table and creates a default row permission for the history table.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - The ADD keyword is optional for:
    - Unnamed PRIMARY KEY constraints
    - Unnamed referential constraints
    - Referential constraints whose name follows the phrase FOREIGN KEY
  - The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause
  - *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword)
  - SET SUMMARY AS can be specified in place of SET MATERIALIZED QUERY AS
  - SET MATERIALIZED QUERY AS DEFINITION ONLY can be specified in place of DROP MATERIALIZED QUERY
  - SET MATERIALIZED QUERY AS (fullselect) can be specified in place of ADD MATERIALIZED QUERY (fullselect)
  - ADD PARTITIONING KEY can be specified in place of ADD DISTRIBUTE BY HASH; the optional USING HASHING clause can also still be specified in this case
  - DROP PARTITIONING KEY can be specified in place of DROP DISTRIBUTION
  - The LONG VARCHAR and LONG VARGRAPHIC data types continue to be supported but are deprecated and not recommended, especially for portable applications
  - A comma can be used to separate multiple options in the *identity-alteration* clause
  - PART can be specified in place of PARTITION
  - VALUES can be specified in place of ENDING AT
  - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER can be specified in place of NO MINVALUE, NO MAXVALUE, NO CYCLE, NO CACHE, and NO ORDER, respectively
  - DROP EXPRESSION can be specified in place of DROP GENERATED to drop the generated expression attribute for a column.
  - DROP IDENTITY can be specified in place of DROP GENERATED to drop the identity attribute for a column.

Examples

- *Example 1:* Add a new column named RATING, which is one character long, to the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
   ADD RATING CHAR(1)
```

- *Example 2:* Add a new column named SITE_NOTES to the PROJECT table. Create SITE_NOTES as a varying-length column with a maximum length of 1000 bytes. The values of the column do not have an associated character set and therefore should not be converted.

```
ALTER TABLE PROJECT
   ADD SITE_NOTES  VARCHAR(1000) FOR BIT DATA
```

- *Example 3:* Assume a table called EQUIPMENT exists defined with the following columns:

```
Column Name        Data Type
EQUIP_NO           INT
EQUIP_DESC         VARCHAR(50)
LOCATION           VARCHAR(50)
EQUIP_OWNER        CHAR(3)
```

  Add a referential constraint to the EQUIPMENT table so that the owner (EQUIP_OWNER) must be a department number (DEPTNO) that is present in the DEPARTMENT table. DEPTNO is the primary key of the DEPARTMENT table. If a department is removed from the DEPARTMENT table, the owner (EQUIP_OWNER) values for all equipment owned by that department should become unassigned (or set to null). Give the constraint the name DEPTQUIP.

```
ALTER TABLE EQUIPMENT
     ADD CONSTRAINT DEPTQUIP
      FOREIGN KEY (EQUIP_OWNER)
        REFERENCES DEPARTMENT
           ON DELETE SET NULL
```

  Also, an additional column is needed to allow the recording of the quantity associated with this equipment record. Unless otherwise specified, the EQUIP_QTY column should have a value of 1 and must never be null.

```
ALTER TABLE EQUIPMENT
  ADD COLUMN EQUIP_QTY
  SMALLINT NOT NULL DEFAULT 1
```

- *Example 4:* Alter table EMPLOYEE. Add the check constraint named REVENUE defined so that each employee must make a total of salary and commission greater than $30,000.

```
ALTER TABLE EMPLOYEE
  ADD CONSTRAINT REVENUE
  CHECK (SALARY + COMM > 30000)
```

- *Example 5:* Alter table EMPLOYEE. Drop the constraint REVENUE which was previously defined.

```
ALTER TABLE EMPLOYEE
  DROP CONSTRAINT REVENUE
```

- *Example 6:* Alter a table to log SQL changes in the default format.

```
ALTER TABLE SALARY1
  DATA CAPTURE NONE
```

- *Example 7:* Alter a table to log SQL changes in an expanded format.

```
ALTER TABLE SALARY2
  DATA CAPTURE CHANGES
```

- *Example 8:* Alter the EMPLOYEE table to add 4 new columns with default values.

```
ALTER TABLE EMPLOYEE
   ADD COLUMN HEIGHT MEASURE   DEFAULT MEASURE(1)
   ADD COLUMN BIRTHDAY BIRTHDATE DEFAULT DATE('01-01-1850')
   ADD COLUMN FLAGS BLOB(1M)  DEFAULT BLOB(X'01')
   ADD COLUMN PHOTO PICTURE   DEFAULT BLOB(X'00')
```

The default values use various function names when specifying the default. Since MEASURE is a distinct type based on INTEGER, the MEASURE function is used. The HEIGHT column default could have been specified without the function since the source type of MEASURE is not BLOB or a datetime data type. Since BIRTHDATE is a distinct type based on DATE, the DATE function is used (BIRTHDATE cannot be used here). For the FLAGS and PHOTO columns the default is specified using the BLOB function even though PHOTO is a distinct type. To specify a default for BIRTHDAY, FLAGS and PHOTO columns, a function must be used because the type is a BLOB or a distinct type sourced on a BLOB or datetime data type.

- *Example 9:* A table called CUSTOMERS is defined with the following columns:

```
Column Name        Data Type
BRANCH_NO          SMALLINT
CUSTOMER_NO        DECIMAL(7)
CUSTOMER_NAME      VARCHAR(50)
```

  In this table, the primary key is made up of the BRANCH_NO and CUSTOMER_NO columns. To distribute the table, you will need to create a distribution key for the table. The table must be defined in a table space on a single-node database partition group. The primary key must be a superset of the distribution key columns: at least one of the columns of the primary key must be used as the distribution key. Make BRANCH_NO the distribution key as follows:

```
ALTER TABLE CUSTOMERS
  ADD DISTRIBUTE BY HASH (BRANCH_NO)
```

- *Example 10:* A remote table EMPLOYEE was created in a federated system using transparent DDL. Alter the remote table EMPLOYEE to add the columns PHONE_NO and WORK_DEPT; also add a primary key on the existing column EMP_NO and the new column WORK_DEPT.

```
ALTER TABLE EMPLOYEE
  ADD COLUMN PHONE_NO CHAR(4) NOT NULL
  ADD COLUMN WORK_DEPT CHAR(3)
  ADD PRIMARY KEY (EMP_NO, WORK_DEPT)
```

- *Example 11:* Alter the DEPARTMENT table to add a functional dependency FD1, then drop the functional dependency FD1 from the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
  ADD CONSTRAINT FD1
    CHECK ( DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED

ALTER TABLE DEPARTMENT
  DROP CHECK FD1
```

- *Example 12:* Change the default value for the WORKDEPT column in the EMPLOYEE table to 123.

```
ALTER TABLE EMPLOYEE
  ALTER COLUMN WORKDEPT
    SET DEFAULT '123'
```

- *Example 13:* Associate the security policy DATA_ACCESS with the table EMPLOYEE.

```
ALTER TABLE EMPLOYEE
  ADD SECURITY POLICY DATA_ACCESS
```

- *Example 14:* Alter the table EMPLOYEE to protect the SALARY column.

```
ALTER TABLE EMPLOYEE
  ALTER COLUMN SALARY
  SECURED WITH EMPLOYEESECLABEL
```

- *Example 15:* Assume that you have a table named SALARY_DATA that is defined with the following columns:

```
Column Name           Data Type
-----------           ---------
EMP_NAME              VARCHAR(50) NOT NULL
EMP_ID                SMALLINT NOT NULL
EMP_POSITION          VARCHAR(100) NOT NULL
SALARY                DECIMAL(5,2)
PROMOTION_DATE        DATE NOT NULL
```

  Change this table to allow salaries to be stored in a DECIMAL(6,2) column, make PROMOTION_DATE an optional field that can be set to the null value, and remove the EMP_POSITION column.

```
ALTER TABLE SALARY_DATA
  ALTER COLUMN SALARY SET DATA TYPE DECIMAL(6,2)
  ALTER COLUMN PROMOTION_DATE DROP NOT NULL
  DROP COLUMN EMP_POSITION
```

- *Example 16:* Add a column named DATE_ADDED to the table BOOKS. The default value for this column is the current timestamp.

```
ALTER TABLE BOOKS
  ADD COLUMN DATE_ADDED TIMESTAMP
  WITH DEFAULT CURRENT TIMESTAMP
```

- *Example 17:* Alter table with label-based access control security attributes into a materialized query table. Base tables tt1 and tt2 exist and were created with the following SQL:

```
CREATE TABLE tt1
   (c1 INT SECURED WITH C, c2 DB2SECURITYLABEL) SECURITY POLICY P;
CREATE TABLE tt2
   (c3 INT SECURED WITH B, c4 DB2SECURITYLABEL) SECURITY POLICY P;
```

  Table tt2 can be altered to be a materialized query table with the following SQL:

```
ALTER TABLE tt2 ADD (SELECT * FROM tt1 WHERE c1 > 10)
   DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

  Table tt2 becomes a materialized query table with the secure policy P. tt2.c3 has security label P.B. tt2.c4 has security label P.C and it is also DB2SECURITYLABEL.

ALTER TABLESPACE

The ALTER TABLESPACE statement is used to modify an existing table space

A table space can be modified in the following ways:

- Add a container to, or drop a container from a DMS table space; that is, a table space created with the MANAGED BY DATABASE option.
- Modify the size of a container in a DMS table space.
- Lower the high water mark for a DMS table space through extent movement.
- Add a container to an SMS table space on a database partition that currently has no containers.
- Modify the PREFETCHSIZE setting for a table space.
- Modify the BUFFERPOOL used for tables in the table space.
- Modify the OVERHEAD setting for a table space.
- Modify the TRANSFERRATE setting for a table space.
- Modify the file system caching policy for a table space.
- Enable or disable auto-resize for a DMS or automatic storage table space.
- Rebalance a regular or large automatic storage table space.
- Modify the DATA TAG setting for a table space.
- Alter a DMS table space to an automatic storage table space.
- Modify the STOGROUP setting associated with a table space.

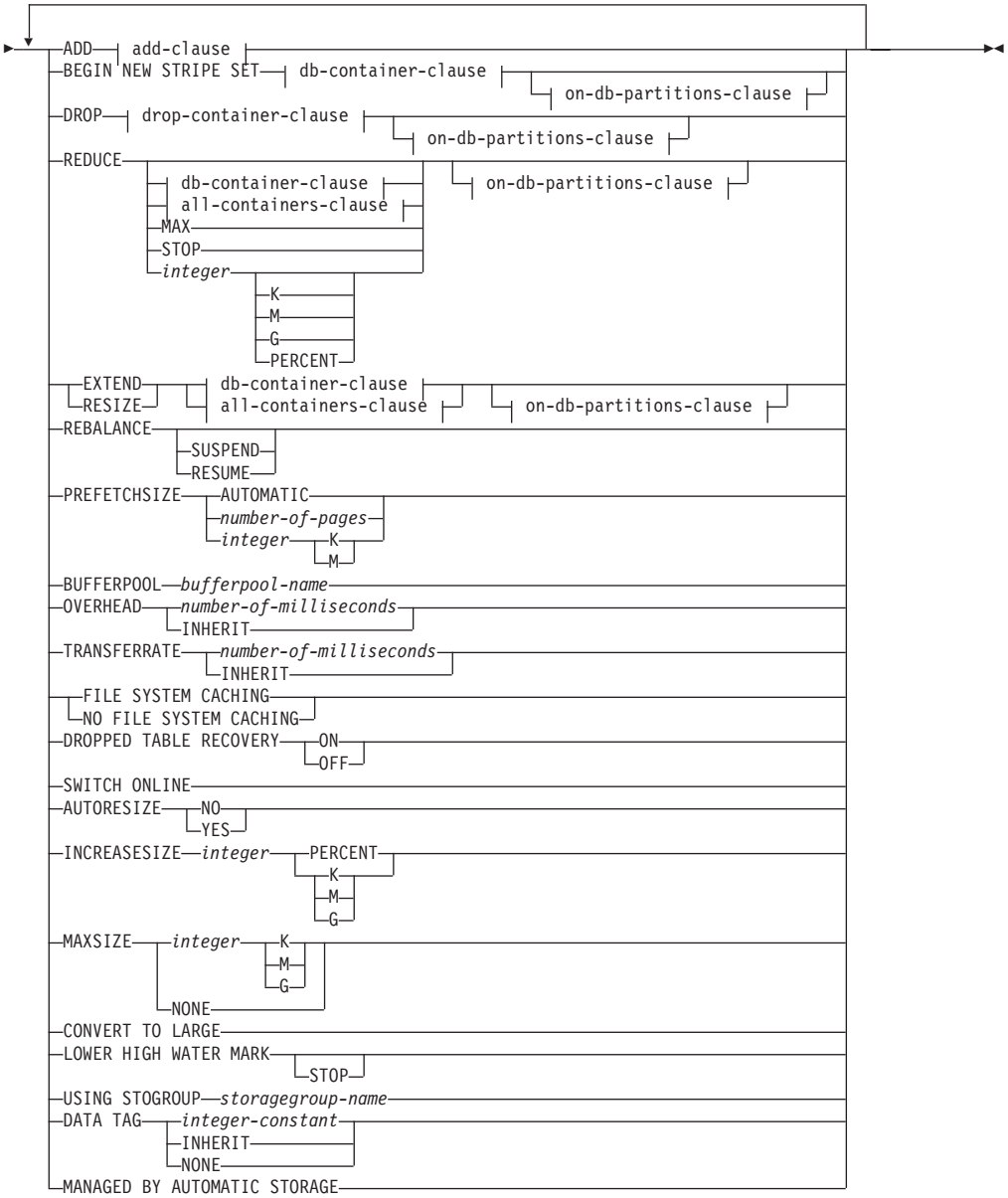
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

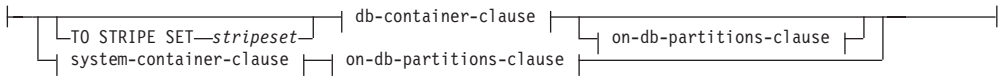
Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

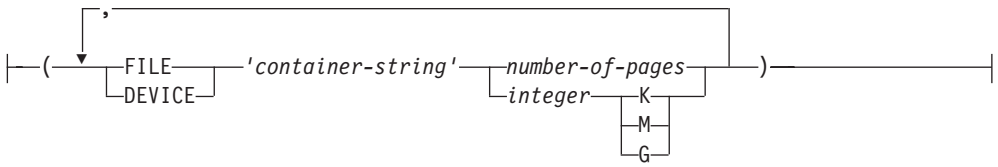
Syntax



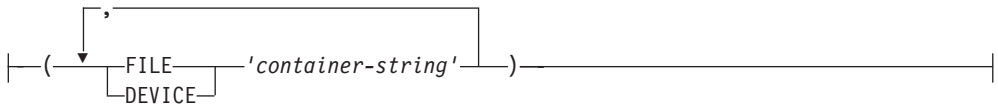
add-clause:



db-container-clause:

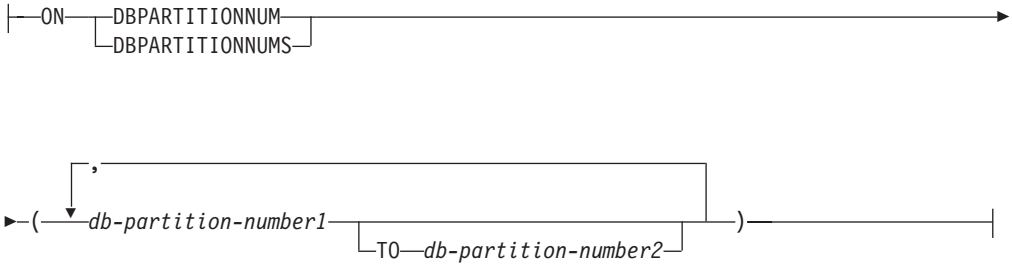


drop-container-clause:

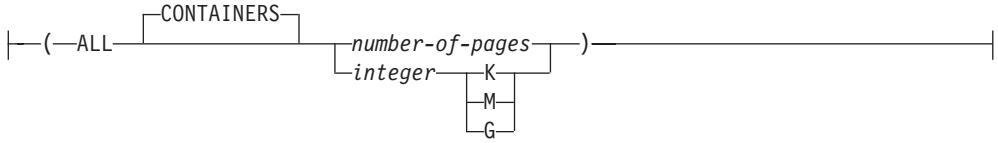


system-container-clause:

on-db-partitions-clause:



all-containers-clause:



Description

tablespace-name
Names the table space. This is a one-part name. It is a long SQL identifier (either ordinary or delimited).

ADD
Specifies that one or more new containers are to be added to the table space.

TO STRIPE SET *stripeset*
Specifies that one or more new containers are to be added to the table space, and that they will be placed into the given stripe set.

BEGIN NEW STRIPE SET
Specifies that a new stripe set is to be created in the table space, and that one or more containers are to be added to this new stripe set. Containers that are subsequently added using the ADD option will be added to this new stripe set unless TO STRIPE SET is specified.

DROP
Specifies that one or more containers are to be dropped from the table space.

REDUCE
For non-automatic storage table spaces, specifies that existing containers are to

be reduced in size. The size specified is the size by which the existing container is decreased. If the *all-containers-clause* is specified, all containers in the table space will decrease by this size. If the reduction in size will result in a table space size that is smaller than the current high water mark, an attempt will be made to reduce the high water mark before attempting to reduce the containers. For non-automatic storage table spaces, the REDUCE clause must be followed by a *db-container-clause* or an *all-containers-clause*.

For automatic storage table spaces, specifies that the current high water mark is to be reduced, if possible, and that the size of the table space is to be reduced to the new high water mark. For automatic storage table spaces, the REDUCE clause must not be followed by a *db-container-clause*, an *all-containers-clause* or an *on-db-partitions-clause*.

Note: The REDUCE option with the MAX, numeric value, PERCENT, or STOP clauses, and the LOWER HIGH WATER MARK option including the STOP clause, are only available for database managed, and automatic storage managed, table spaces with the reclaimable storage attribute. Moreover, these options must be specified and run without any other options, including each other.

The **MAX**, **STOP**, *integer* **[K | M | G]**, or *integer* **PERCENT** clause takes effect when the statement is processed and is not rolled back if the unit of work, in which the statement is executed, is rolled back.

db-container-clause
Adds one or more containers to a DMS table space. The table space must identify a DMS table space that already exists at the application server.

all-containers-clause
Extends, reduces, or resizes all of the containers in a DMS table space. The table space must identify a DMS table space that already exists at the application server.

MAX
For automatic storage table spaces with reclaimable storage, specifies that the maximum number of extents should be moved to the beginning of the table space to lower the high water mark. Additionally, the size of the table space will be reduced to the new high water mark. This does not apply to non-automatic storage table spaces.

STOP
For automatic storage table spaces with reclaimable storage, interrupts the extent movement operation if in progress. This option is not available for non-automatic storage table spaces.

integer **[K | M | G] or** *integer* **PERCENT**
For automatic storage table spaces with reclaimable storage, specifies the numeric value by which the table space is to be reduced through extent movement. The value can be expressed in several ways:

- An integer specified without K, M, G, or PERCENT indicates that the numeric value is the number of pages by which the table space is to be reduced.
- An integer specified with K, M, or G indicates the reduction size in kilobytes, megabytes, or gigabytes, respectively. The value is first converted from bytes to number of pages based on the page size of the table space.
- An integer specified with PERCENT indicates the number of extents to move, as a percentage of the current size of the table space.

Once extent movement is complete, the table space size is reduced to the new high water mark. This option is not available for non-automatic storage table spaces.

on-db-partitions-clause
Specifies one or more database partitions for the corresponding container operations.

EXTEND
Specifies that existing containers are to be increased in size. The size specified is the size by which the existing container is increased. If the *all-containers-clause* is specified, all containers in the table space will increase by this size.

RESIZE
Specifies that the size of existing containers is to be changed. The size specified is the new size for the container. If the *all-containers-clause* is specified, all containers in the table space will be changed to this size. If the operation affects more than one container, these containers must all either increase in size, or decrease in size. It is not possible to increase some while decreasing others (SQLSTATE 429BC).

db-container-clause
Adds one or more containers to a DMS table space. The table space must identify a DMS table space that already exists at the application server.

drop-container-clause
Drops one or more containers from a DMS table space. The table space must identify a DMS table space that already exists at the application server.

system-container-clause
Adds one or more containers to an SMS table space on the specified database partitions. The table space must identify an SMS table space that already exists at the application server. There must not be any containers on the specified database partitions for the table space (SQLSTATE 42921).

on-db-partitions-clause
Specifies one or more database partitions for the corresponding container operations.

all-containers-clause
Extends, reduces, or resizes all of the containers in a DMS table space. The table space must identify a DMS table space that already exists at the application server.

REBALANCE
For regular and large automatic storage table spaces, initiates the creation of containers on recently added storage paths, the drop of containers from storage paths that are in the "Drop Pending" state, or both. During the rebalance, data is moved into containers on new paths, and moved out of containers on dropped paths. The rebalance runs asynchronously in the background and does not affect the availability of data.

Note: The **SUSPEND** or **RESUME** clause takes effect when the statement is processed and is not rolled back if the unit of work, in which the statement is executed, is rolled back.

SUSPEND
Suspends the active rebalance operation on the specified table space. If there is no active rebalance operation, no action is taken and success is returned. The suspend state is persistent and if the database is deactivated

while the rebalance is suspended, then upon database activation the rebalance operation is restarted from the suspended state. Suspending a rebalance operation when it is already suspended has no effect and success is returned.

RESUME
Resumes a previously suspended rebalance operation. If there is no active rebalance operation, no action is taken and success is returned. If the rebalance is PAUSED because of an online backup operation, then the table space rebalance is taken out of the suspended state but remains paused until the online backup is completed.

PREFETCHSIZE
Specifies to read in data needed by a query before it being referenced by the query, so that the query need not wait for I/O to be performed.

AUTOMATIC
Specifies that the prefetch size of a table space is to be updated automatically; that is, the prefetch size will be managed by DB2 database manager.

A DB2 database will update the prefetch size automatically whenever the number of containers in a table space changes (following successful execution of an ALTER TABLESPACE statement that adds or drops one or more containers). The prefetch size is also automatically updated at database startup.

Automatic updating of the prefetch size can be turned off by specifying a numeric value in the PREFETCHSIZE clause.

number-of-pages
Specifies the number of PAGESIZE pages that will be read from the table space when data prefetching is being performed. The maximum value is 32767.

integer **K | M**
Specifies the prefetch size value as an integer value followed by K (for kilobytes) or M (for megabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the number of pages value for prefetch size.

BUFFERPOOL *bufferpool-name*
The name of the buffer pool used for tables in this table space. The buffer pool must currently exist in the database (SQLSTATE 42704). The database partition group of the table space must be defined for the bufferpool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds* **or OVERHEAD INHERIT**
Specifies the I/O controller overhead and disk seek and latency time. This value is used to determine the cost of I/O during query optimization.

number-of-milliseconds
Any numeric literal (integer, decimal, or floating point) that specifies the I/O controller overhead and disk seek and latency time, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers.

INHERIT
If INHERIT is specified, the table space must be defined using automatic storage and the OVERHEAD is dynamically inherited from the storage group. INHERIT cannot be specified if the table space is not defined using automatic storage (SQLSTATE 42858). If the OVERHEAD is set to

undefined for the storage group and you set OVERHEAD to INHERIT, the database creation default will be used.

For a database that was created in DB2 V10.1 or later, the default I/O controller overhead and disk seek and latency time is 6.725 milliseconds.

For a database that was upgraded from a previous version of DB2 to DB2 V10.1 or later, the default I/O controller overhead and disk seek and latency time is as follows:

- 7.5 milliseconds for a database created in DB2 version 9.1 or higher
- 12.67 milliseconds for databases created between DB2 version 8.2 and DB2 version 9.1
- 24.1 milliseconds for DB2 versions previous to 8.2

TRANSFERRATE *number-of-milliseconds* **or TRANSFERRATE INHERIT**
Specifies the time to read one page into memory. This value is used to determine the cost of I/O during query optimization.

number-of-milliseconds
Any numeric literal (integer, decimal, or floating point) that specifies the time to read one page (4K or 8K) into memory, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers.

INHERIT
If INHERIT is specified, the table space must be defined using automatic storage and the TRANSFERRATE is dynamically inherited from the storage group. INHERIT cannot be specified if the table space is not defined using automatic storage (SQLSTATE 42858). If the DEVICE READ RATE of the storage group is set to undefined and the user sets TRANSFERRATE to INHERIT, the database creation default will be used.

When an automatic storage table space inherits the TRANSFERRATE setting from the storage group it is using, the DEVICE READ RATE of the storage group, which is in megabytes per second, is converted into milliseconds per page read accounting for the table space's PAGESIZE setting of the table space. The conversion formula follows:

TRANSFERRATE = (1 / DEVICE READ RATE) * 1000 / 1024000 * PAGESIZE

For a database that was created in DB2 V10.1 or later, the default time to read one page into memory for 4 KB PAGESIZE table space is 0.04 milliseconds.

For a database that was upgraded from a previous version of DB2 to DB2 V10.1 or later, the default time to read one page into memory is as follows:

- 0.06 milliseconds for a database created in DB2 version 9.1 or higher
- 0.18 milliseconds for databases created between DB2 version 8.2 or version 9.1
- 0.9 milliseconds for DB2 versions previous to 8.2

FILE SYSTEM CACHING or NO FILE SYSTEM CACHING
Specifies whether or not I/O operations will be cached at the file system level. Connections to the database must be terminated before a new caching policy takes effect. Note that I/O access to long or LOB data is buffered for both SMS and DMS containers.

FILE SYSTEM CACHING
All I/O operations in the target table space will be cached at the file system level.

NO FILE SYSTEM CACHING
All I/O operations will bypass the file system level cache.

DROPPED TABLE RECOVERY
Specifies whether or not tables that have been dropped from *tablespace-name* can be recovered using the **RECOVER DROPPED TABLE ON** option of the **ROLLFORWARD DATABASE** command. For partitioned tables, dropped table recovery is always on, even if dropped table recovery is turned off for non-partitioned tables in one or more table spaces.

ON Specifies that dropped tables can be recovered.

OFF
Specifies that dropped tables cannot be recovered.

SWITCH ONLINE
Specifies that table spaces in OFFLINE state are to be brought online if their containers have become accessible. If the containers are not accessible, an error is returned (SQLSTATE 57048).

AUTORESIZE
Specifies whether or not the auto-resize capability of a database managed space (DMS) table space or an automatic storage table space is to be enabled. Auto-resizable table spaces automatically increase in size when they become full.

NO Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be disabled. If the auto-resize capability is disabled, any values that have been previously specified for INCREASESIZE or MAXSIZE will not be kept.

YES
Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled.

INCREASESIZE *integer* **PERCENT or INCREASESIZE** *integer* **K | M | G**
Specifies the amount, per database partition, by which a table space that is enabled for auto-resize will automatically be increased when the table space is full, and a request for space has been made. The integer value must be followed by:

- PERCENT to specify the amount as a percentage of the table space size at the time that a request for space is made. When PERCENT is specified, the integer value must be between 0 and 100 (SQLSTATE 42615).
- K (for kilobytes), M (for megabytes), or G (for gigabytes) to specify the amount in bytes

Note that the actual value used might be slightly smaller or larger than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

MAXSIZE *integer* **K | M | G or MAXSIZE NONE**
Specifies the maximum size to which a table space that is enabled for auto-resize can automatically be increased.

integer
Specifies a hard limit on the size, per database partition, to which a DMS table space or an automatic storage table space can automatically be

increased. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

NONE
Specifies that the table space is to be allowed to grow to file system capacity, or to the maximum table space size (described in "SQL and XML limits").

CONVERT TO LARGE
Modifies an existing regular DMS table space to be a large DMS table space. The table space and its contents are locked during conversion. This option can only be used on regular DMS table spaces. If an SMS table space, a temporary table space, or the system catalog table space is specified, an error is returned (SQLSTATE 560CF). You cannot convert a table space that contains a data partition of a partitioned table that has data partitions in another table space (SQLSTATE 560CF). Conversion cannot be reversed after being committed. If tables in the table space are defined with DATA CAPTURE CHANGES, consider the storage and capacity limits of the target table and table space.

LOWER HIGH WATER MARK
For both automatic storage and non-automatic storage table spaces with reclaimable storage, triggers the extent movement operation to move the maximum number of extents lower in the table space. Although the high water mark is lowered, the size of the table space is not reduced. This must be followed by an ALTER TABLESPACE REDUCE for automatic storage table spaces or ALTER TABLESPACE REDUCE with the *db-container-clause* or *all-containers-clause* for non-automatic storage table spaces.

Note: The LOWER HIGH WATER MARK option including the STOP clause, and the REDUCE option with the MAX, numeric value, PERCENT, or STOP clauses, are only available for database managed and automatic storage managed table spaces with the reclaimable storage attribute. Moreover, these options must be specified and run without any other options, including each other.

Note: This clause takes effect when the statement is processed and is not rolled back if the unit of work, in which the statement is executed, is rolled back.

STOP
For both automatic storage and non-automatic storage table spaces with reclaimable storage, interrupts the extent movement operation if in progress.

USING STOGROUP
Associates a table space with a different storage group. The data associated with the table space will be moved from its current storage group to the specified storage group. This clause only applies to automatic storage table spaces unless specified with the MANAGED BY AUTOMATIC STORAGE clause (SQLSTATE 42858).

For automatic storage table spaces, an implicit REBALANCE is started at commit time. For a database managed table space being converted to automatic storage managed, an explicit REBALANCE statement is required.

In a partitioned database environment, to alter the storage group association of a table space, the table space must be defined using automatic storage on all

database partitions. If the table space on any database partition is not defined using automatic storage, this command will fail unless specified with the MANAGED BY AUTOMATIC STORAGE clause (SQLSTATE 42858). However, it is not required that a table space have the same storage group association on all database partitions for this command to succeed in moving the table space on all database partitions.

storagegroup-name
Identifies the storage group in which table space data will be stored. *storagegroup-name* must identify a storage group that exists at the current server (SQLSTATE 42704). This is a one-part name.

DATA TAG *integer-constant*, **DATA TAG INHERIT or DATA TAG NONE**
Specifies a tag for the data in the table space. This value can be used as part of a WLM configuration in a work class definition or referenced within a threshold definition; for more information refer to the CREATE WORK CLASS SET, ALTER WORK CLASS SET, CREATE THRESHOLD, and ALTER THRESHOLD statements. This clause cannot be specified for USER or SYSTEM TEMPORARY table spaces or for the catalog table space (SQLSTATE 42858).

integer-constant
Valid values for *integer-constant* are integers from 1 to 9. If an *integer-constant* is specified and there is an associated storage group, the data tag specified for the table space will override any data tag value specified for the associated storage group.

INHERIT
If INHERIT is specified, the table space must be defined using automatic storage and the DATA TAG is dynamically inherited from the storage group. INHERIT cannot be specified if the table space is not defined using automatic storage (SQLSTATE 42858).

NONE
If NONE is specified, there is no data tag.

MANAGED BY AUTOMATIC STORAGE
Enables automatic storage for a database managed (DMS) table space. Once automatic storage is enabled, no further container operations can be executed on the table space. The table space being converted cannot be using RAW (DEVICE) containers.

If the USING STOGROUP clause is not included when converting from a DMS table space to an automatic storage table space then the default storage group is specified.

Rules

- The BEGIN NEW STRIPE SET clause cannot be specified in the same statement as ADD, DROP, EXTEND, REDUCE, and RESIZE, unless those clauses are being directed to different database partitions (SQLSTATE 429BC).
- The stripe set value specified with the TO STRIPE SET clause must be within the valid range for the table space being altered (SQLSTATE 42615).
- When adding or removing space from the table space, the following rules must be followed:
  - EXTEND and RESIZE can be used in the same statement, provided that the size of each container is increasing (SQLSTATE 429BC).
  - REDUCE and RESIZE can be used in the same statement, provided that the size of each container is decreasing (SQLSTATE 429BC).

  - EXTEND and REDUCE cannot be used in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
  - ADD cannot be used with REDUCE or DROP in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
  - DROP cannot be used with EXTEND or ADD in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
- The AUTORESIZE, INCREASESIZE, or MAXSIZE clause cannot be specified for system managed space (SMS) table spaces, temporary table spaces that were created using automatic storage, or DMS table spaces that are defined to use raw device containers (SQLSTATE 42601).
- The INCREASESIZE or MAXSIZE clause cannot be specified if the table space is not auto-resizable (SQLSTATE 42601).
- When specifying a new maximum size for a table space, the value must be larger than the current size on each database partition (SQLSTATE 560B0).
- Container operations (ADD, EXTEND, RESIZE, DROP, or BEGIN NEW STRIPE SET) cannot be performed on automatic storage table spaces, because the database manager is controlling the space management of such table spaces (SQLSTATE 42858).
- Raw device containers cannot be added to an auto-resizable DMS table space (SQLSTATE 42601).
- The CONVERT TO LARGE clause cannot be specified in the same statement as any other clause (SQLSTATE 429BC).
- The REBALANCE clause cannot be specified with any other clause (SQLSTATE 429BC).
- The REBALANCE clause is only valid for regular and large automatic storage table spaces (SQLSTATE 42601). Temporary automatic storage table spaces should be dropped and recreated to take advantage of recently added storage paths or to have their containers removed from storage paths being dropped.
- Container operations and the REBALANCE clause cannot be specified if the table space is in the “DMS rebalancer is active” state (SQLSTATE 55041).
- The USING STOGROUP clause cannot be specified for temporary table spaces (SQLSTATE 42858).
- The following clauses are not supported in DB2 pureScale environments:
  - ADD *db-container-clause*
  - BEGIN NEW STRIPE SET *db-container-clause*
  - DROP *db-container-clause*
  - LOWER HIGH WATER MARK
  - LOWER HIGH WATER MARK STOP
  - REDUCE, unless it is specified without any of its optional elements
  - RESIZE *db-container-clause*
  - USING STOGROUP
- The ADD, DROP, RESIZE, EXTEND, REDUCE, LOWER HIGH WATER MARK, and BEGIN_STRIPE_SET clauses cannot be used in conjunction with the MANAGED BY AUTOMATIC STORAGE clause or the USING STOGROUP clause (SQLSTATE 429BC).
- The USING STOGROUP clause cannot be specified if the table space is in the "rebalancer is active" state (SQLSTATE 55041).
- ***Container size limit***: In DMS table spaces, a container must be at least two times the extent size pages in length (SQLSTATE 54039). The maximum size of a container is operating system dependent.

- ***Container definition length limit***: Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container definitions for the table space cannot exceed 208 kilobytes (SQLSTATE 54034).

Notes

- Default container operations are container operations that are specified in the ALTER TABLESPACE statement, but that are not explicitly directed to a specific database partition. These container operations are sent to any database partition that is not listed in the statement. If these default container operations are not sent to any database partition, because all database partitions are explicitly mentioned for a container operation, a warning is returned (SQLSTATE 01589).
- Once space has been added or removed from a table space, and the transaction is committed, the contents of the table space may be rebalanced across the containers. Access to the table space is not restricted during rebalancing.
- If the table space is in OFFLINE state and the containers have become accessible, the user can disconnect all applications and connect to the database again to bring the table space out of OFFLINE state. Alternatively, SWITCH ONLINE option can bring the table space up (out of OFFLINE) while the rest of the database is still up and being used.
- If adding more than one container to a table space, it is recommended that they be added in the same statement so that the cost of rebalancing is incurred only once. An attempt to add containers to the same table space in separate ALTER TABLESPACE statements within a single transaction will result in an error (SQLSTATE 55041).
- Any attempts to extend, reduce, resize, or drop containers that do not exist will raise an error (SQLSTATE 428B2).
- When extending, reducing, or resizing a container, the container type must match the type that was used when the container was created (SQLSTATE 428B2).
- An attempt to change container sizes in the same table space, using separate ALTER TABLESPACE statements but within a single transaction, will raise an error (SQLSTATE 55041).
- In a partitioned database if more than one database partition resides on the same physical node, the same device or specific path cannot be specified for such database partitions (SQLSTATE 42730). For this environment, either specify a unique *container-string* for each database partition or use a relative path name.
- Although the table space definition is transactional and the changes to the table space definition are reflected in the catalog tables on commit, the buffer pool with the new definition cannot be used until the next time the database is started. The buffer pool in use, when the ALTER TABLESPACE statement was issued, will continue to be used in the interim.
- The REDUCE, RESIZE, or DROP option attempts to free unused extents, if necessary, for DMS table spaces, and the REDUCE option attempts to free unused extents for automatic storage table spaces. The removal of unused extents allows the table space high water mark to be reduced to a value that accurately represents the amount of space used, which, in turn, enables larger reductions in table space size.
- ***Conversion to large DMS table spaces***: After conversion, it is recommended that you issue the COMMIT statement and then increase the storage capacity of the table space.
  - If the table space is enabled for auto-resize, the MAXSIZE table space attribute should be increased, unless it is already set to NONE.

  - If the table space is not enabled for auto-resize:
    - Enable auto-resize by issuing the ALTER TABLESPACE statement with the AUTORESIZE YES option, or
    - Add more storage by adding stripe sets, extending the size of existing containers, or both

  Indexes for tables in a converted table space must be reorganized or rebuilt before they can support large record identifiers (RIDs).
  - The indexes can be rebuilt using the **REORG INDEXES ALL** command with the REBUILD option. Specify the **ALLOW NO ACCESS** option for partitioned tables.
  - Alternatively, the tables can be reorganized (not INPLACE), which will rebuild all indexes and enable the tables to support more than 255 rows per page.

  To determine which tables do not yet support large RIDs, use the ADMIN_GET_TAB_INFO table function.
- The rebalance of an automatic storage table space that has containers on a storage path in the "Drop Pending" state will drop those containers. New containers may need to be created to hold the data being moved off the dropped containers. There must be sufficient free space on the other storage paths in the database to allow those containers to be created, otherwise an error is returned SQLSTATE 57011. The actual amount of free space required depends on many factors, including the location of the high-water mark extent and the stripe sets being altered. However, to ensure that the operation will be successful, there should be at least enough free space on the remaining storage paths as there is space being consumed by the containers being dropped.
- If the REBALANCE clause is specified but the data server determines that there is no need to create new containers or drop existing ones, a rebalance does not occur and the statement succeeds with a warning (SQLSTATE 01690).
- In addition to adding containers on recently added paths, the REBALANCE operation may also be used to add containers on existing storage paths. Each stripe set in the table space is examined and storage paths that are not in use by a particular stripe set are identified. For each storage path identified, if there is sufficient free space on it then a new container will be created. The container will have the same size as the other containers in the stripe set. This would be beneficial if a given storage path ran out of space, table spaces stopped using it (by creating stripe sets on the other paths), and more storage was given to the path. In this case, no new paths have been added, but the rebalance will attempt to include that storage path in stripe sets where it wasn't included before.
- Auto-resize can still occur while a rebalance of an automatic storage table space is in progress.
- When a DMS table space is enabled for automatic storage by the MANAGED BY AUTOMATIC STORAGE clause, that table space will have one or more stripe sets of user-defined (non-automatic storage) containers and one or more stripe sets of automatic storage containers. Rebalancing the table space (using the REBALANCE clause) removes all of the user-defined containers. The database manager might extend existing automatic storage containers or create new automatic storage containers to hold the data being moved from the user-defined containers.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NODE can be specified in place of DBPARTITIONNUM
  - NODES can be specified in place of DBPARTITIONNUMS

Examples

- *Example 1:* Add a device to the PAYROLL table space.

```
ALTER TABLESPACE PAYROLL
  ADD (DEVICE '/dev/rhdisk9' 10000)
```

- *Example 2:* Change the prefetch size and I/O overhead for the ACCOUNTING table space.

```
ALTER TABLESPACE ACCOUNTING
  PREFETCHSIZE 64
  OVERHEAD 19.3
```

- *Example 3:* Create a table space TS1, then resize the containers so that all of the containers have 2000 pages. (Three different ALTER TABLESPACE statements that will accomplish this resizing are shown.)

```
CREATE TABLESPACE TS1
  MANAGED BY DATABASE
  USING (FILE '/conts/cont0' 1000,
         DEVICE '/dev/rcont1' 500,
         FILE 'cont2' 700)
ALTER TABLESPACE TS1
  RESIZE (FILE '/conts/cont0' 2000,
          DEVICE '/dev/rcont1' 2000,
          FILE 'cont2' 2000)
```

  OR

```
ALTER TABLESPACE TS1
  RESIZE (ALL 2000)
```

  OR

```
ALTER TABLESPACE TS1
  EXTEND (FILE '/conts/cont0' 1000,
          DEVICE '/dev/rcont1' 1500,
          FILE 'cont2' 1300)
```

- *Example 4:* Extend all of the containers in the DATA_TS table space by 1000 pages.

```
ALTER TABLESPACE DATA_TS
  EXTEND (ALL 1000)
```

- *Example 5:* Resize all of the containers in the INDEX_TS table space to 100 megabytes (MB).

```
ALTER TABLESPACE INDEX_TS
  RESIZE (ALL 100 M)
```

- *Example 6:* Add three new containers. Extend the first container, and resize the second.

```
ALTER TABLESPACE TS0
  ADD (FILE 'cont2' 2000, FILE 'cont3' 2000)
  ADD (FILE 'cont4' 2000)
  EXTEND (FILE 'cont0' 100)
  RESIZE (FILE 'cont1' 3000)
```

- *Example 7:* Table space TSO exists on database partitions 0, 1 and 2. Add a new container to database partition 0. Extend all of the containers on database partition 1. Resize a container on all database partitions other than the ones that were explicitly specified (that is, database partitions 0 and 1).

```
ALTER TABLESPACE TS0
  ADD (FILE 'A' 200) ON DBPARTITIONNUM (0)
  EXTEND (ALL 200) ON DBPARTITIONNUM (1)
  RESIZE (FILE 'B' 500)
```

The RESIZE clause is the default container clause in this example, and will be executed on database partition 2, because other operations are being explicitly sent to database partitions 0 and 1. If, however, there had only been these two database partitions, the statement would have succeeded, but returned a warning (SQL1758W) that default containers had been specified but not used.

- *Example 8:* Enable the auto-resize option for table space DMS_TS1, and set its maximum size to 256 megabytes.

```
ALTER TABLESPACE DMS_TS1
  AUTORESIZE YES MAXSIZE 256 M
```

- *Example 9:* Enable the auto-resize option for table space AUTOSTORE1, and change its growth rate to 5%.

```
ALTER TABLESPACE AUTOSTORE1
  AUTORESIZE YES INCREASESIZE 5 PERCENT
```

- *Example 10:* Change the growth rate for an auto-resizable table space named MY_TS to 512 kilobytes, and set its maximum size to be as large as possible.

```
ALTER TABLESPACE MY_TS
  INCREASESIZE 512 K MAXSIZE NONE
```

- *Example 11:* Enable automatic storage for database managed table space DMS_TS10 and have it use storage group sg_3.

```
ALTER TABLESPACE DMS_TS10
  MANAGED BY AUTOMATIC STORAGE
  USING STOGROUP sg_3
```

- *Example 12*: An ALTER DATABASE statement removed the paths `/db2/filesystem1` and `/db2/filesystem2` from the currently connected database. The table spaces named PRODTS1, PRODTS2, and PRODTS3 were the only table spaces using the removed paths. Rebalance these table spaces. Three ALTER TABLESPACE statements must be used.

```
ALTER TABLESPACE PRODTS1 REBALANCE
ALTER TABLESPACE PRODTS2 REBALANCE
ALTER TABLESPACE PRODTS3 REBALANCE
```

- *Example 13:* Enable automatic storage for database managed table space DATA1 and remove all of the existing non-automatic storage containers from the table space. The first statement must be committed before the second statement can be run.

```
ALTER TABLESPACE DATA1 MANAGED BY AUTOMATIC STORAGE
ALTER TABLESPACE DATA1 REBALANCE
```

- *Example 14:* Trigger extent movement for an automatic storage table space with reclaimable storage attribute, to reduce the size of the containers by 10MB.

```
ALTER TABLESPACE DMS_TS1 REDUCE 10 M
```

- *Example 15:* Trigger extent movement for a non-automatic storage table space with reclaimable storage attribute and subsequently reduce the size of each container by 10MB.

```
ALTER TABLESPACE TBSP1 LOWER HIGH WATER MARK
ALTER TABLESPACE TBSP1 REDUCE (ALL CONTAINERS 10 M)
```

ALTER THRESHOLD

The ALTER THRESHOLD statement alters the definition of a threshold.

Invocation

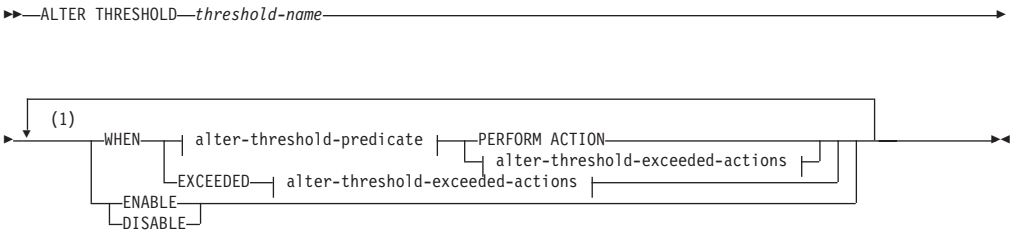
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

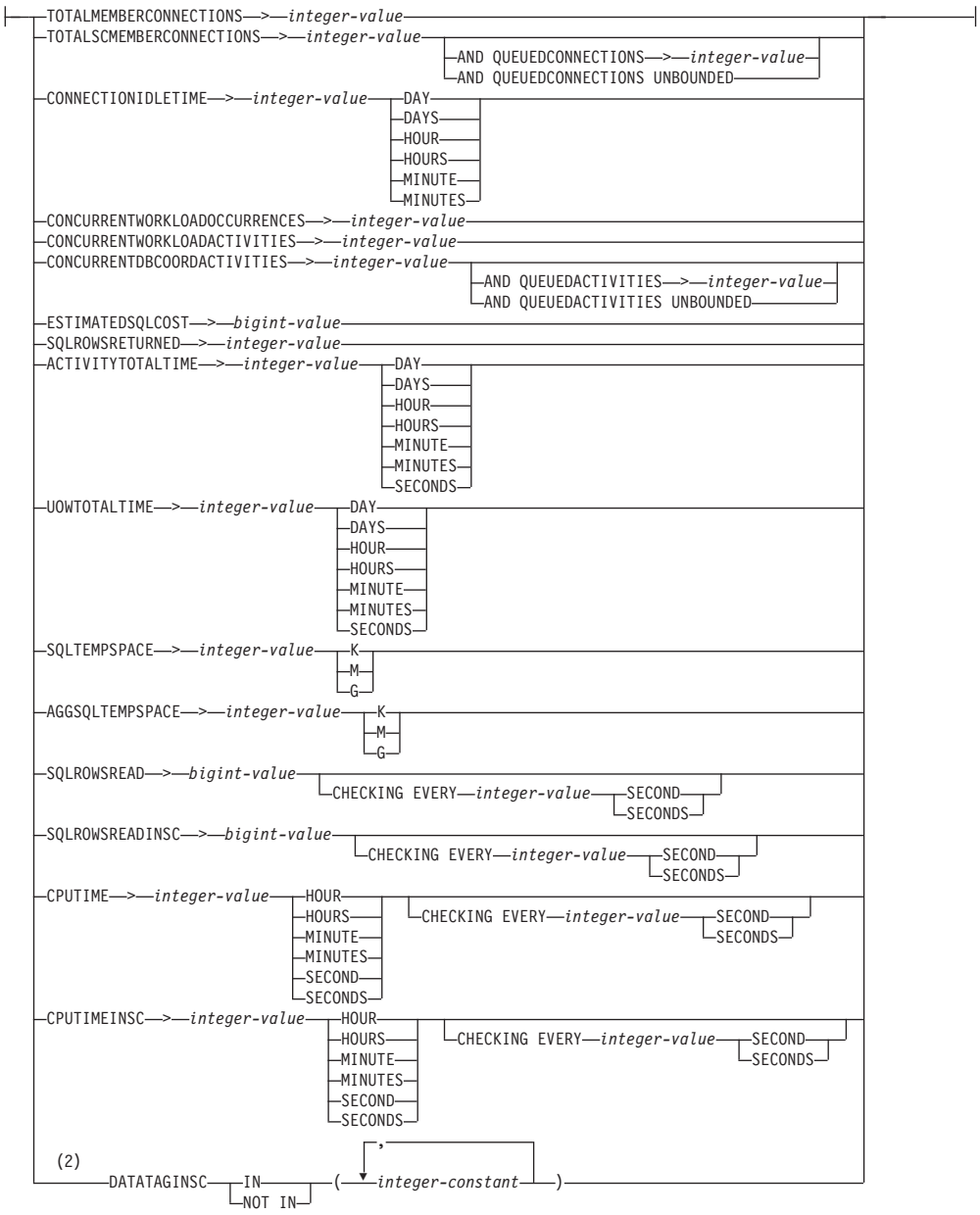
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- SQLADM authority, only if every alteration clause is a COLLECT clause
- WLMADM authority
- DBADM authority

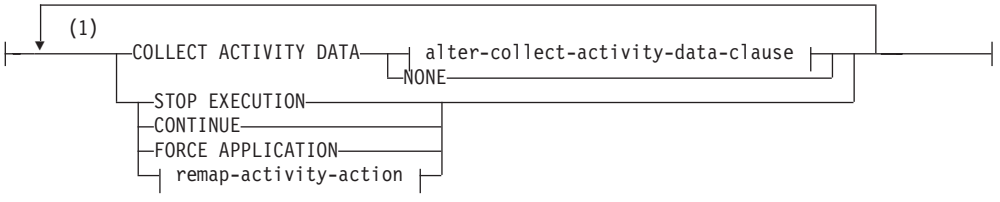
Syntax



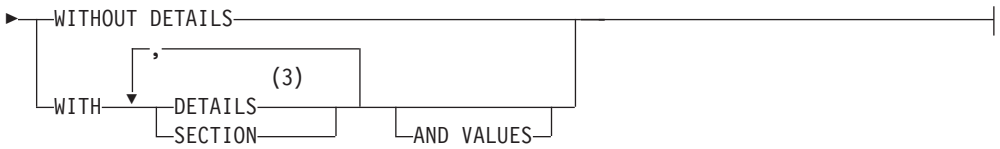
alter-threshold-predicate:



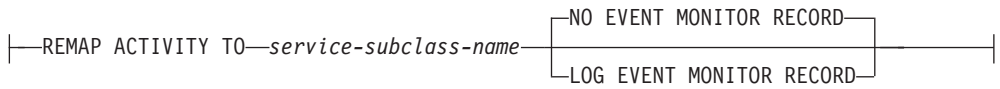
alter-threshold-exceeded-actions:



alter-collect-activity-data-clause:



remap-activity-action:



Notes:

1 The same clause must not be specified more than once.

2 Each data tag value can be specified only once.

3 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

Description

threshold-name
Identifies the threshold to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must uniquely identify an existing threshold at the current server (SQLSTATE 42704).

WHEN *alter-threshold-predicate* **or WHEN EXCEEDED**
Replaces the existing upper bound value in the threshold predicate condition with a new upper bound value. The condition of the threshold cannot be changed to a different one.

PERFORM ACTION
When altering the value of the threshold predicate condition, specifies that the threshold exceeded action is not changed.

EXCEEDED
Specifies to keep the same threshold predicate that was specified originally for this altered threshold.

alter-threshold-predicate

TOTALMEMBERCONNECTIONS > *integer-value*
This condition defines an upper bound on the number of coordinator connections that can run concurrently on a member. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that any new coordinator connection will be prevented from connecting. All currently running or queued connections will continue.

TOTALSCMEMBERCONNECTIONS > *integer-value*
This condition defines an upper bound on the number of coordinator connections that can run concurrently on a member in a specific service superclass. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that any new connection will be prevented from joining the service class. All currently running or queued connections will continue.

AND QUEUEDCONNECTIONS > *integer-value* **or AND QUEUEDCONNECTIONS UNBOUNDED**
Specifies a queue size for when the maximum number of coordinator connections is exceeded. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that no coordinator connections are queued. Specifying UNBOUNDED will queue every connection that exceeds the specified maximum number of coordinator connections, and the *threshold-exceeded-actions* will never be executed.

CONNECTIONIDLETIME > *integer-value* **DAY | DAYS | HOUR | HOURS | MINUTE | MINUTES**
This condition defines an upper bound for the amount of time the database manager will allow a connection to remain idle. This value can be any positive integer (not zero) (SQLSTATE 42820). Use a valid duration keyword to specify an appropriate unit of time for *integer-value*. This condition is enforced at the coordinator member.

If you specify the STOP EXECUTION action with CONNECTIONIDLETIME thresholds, the connection for the application is dropped when the threshold is exceeded. Any subsequent attempt by the application to access the data server will not receive SQLSTATE 5U026 since the application is no longer connected to the data server..

The maximum value for this threshold is 2 147 483 640 seconds. Any value specified that has a seconds equivalent larger than 2 147 483 640 seconds will be set to this number of seconds.

CONCURRENTWORKLOADOCCURRENCES > *integer-value*
This condition defines an upper bound on the number of concurrent occurrences for the workload on each member. This value can be any positive integer (not zero) (SQLSTATE 42820).

CONCURRENTWORKLOADACTIVITIES > *integer-value*
This condition defines an upper bound on the number of concurrent coordinator activities and nested activities for the workload on each member. This value can be any positive integer (not zero) (SQLSTATE 42820).

Each nested activity must satisfy the following conditions:

- It must be a recognized coordinator activity. Any nested coordinator activity that does not fall within the recognized types of activities will not be counted. Similarly, nested subagent activities, such as remote node requests, are not counted.
- It must be directly invoked from user logic, such as a user-written procedure issuing SQL statements.

Consequently, nested coordinator activities that were automatically started under the invocation of a DB2 utility or routines in the SYSIBM, SYSFUN, or SYSPROC schemas are not counted toward the upper bound specified by this threshold.

Internal SQL activities, such as those generated by the setting of a constraint or the refreshing of a materialized query table, are also not counted by this threshold, because they are initiated by the database manager and not directly invoked by user logic.

CONCURRENTDBCOORDACTIVITIES > *integer-value*
This condition defines an upper bound on the number of recognized database coordinator activities that can run concurrently on all members in the specified domain. This value can be zero or any positive integer

(SQLSTATE 42820). A value of zero means that any new database coordinator activities will be prevented from executing. All currently running or queued database coordinator activities will continue. All activities are tracked by this condition, except for the following items:

- CALL statements are not controlled by this threshold, but all nested child activities started within the called routine are under this threshold's control. Anonymous blocks and autonomous routines are classified as CALL statements.
- User-defined functions are controlled by this threshold, but child activities nested in a user-defined function are not controlled. If an autonomous routine is called from within a user defined function, neither the autonomous routine nor any child activities of the autonomous routine are under threshold control.
- Trigger actions that invoke CALL statements and the child activities of these CALL statements are not controlled by this threshold. INSERT, UPDATE, or DELETE statements that can cause a trigger to activate continue to be under threshold control.

Important: Before using CONCURRENTDBCOORDACTIVITIES thresholds, be sure to become familiar with the effects that they can have on the database system. For more information, see the "CONCURRENTDBCOORDACTIVITIES threshold" topic.

AND QUEUEDACTIVITIES > *integer-value* **or AND QUEUEDACTIVITIES UNBOUNDED**

Specifies a queue size for when the maximum number of database coordinator activities is exceeded. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that no database coordinator activities are queued. Specifying UNBOUNDED will queue every database coordinator activity that exceeds the specified maximum number of database coordinator activities, and the *threshold-exceeded-actions* will never be executed.

Note: If a threshold action of CONTINUE is specified for a queuing threshold, it effectively makes the size of the queue unbounded, regardless of any hard value specified for the queue size.

ESTIMATEDSQLCOST > *bigint-value*

This condition defines an upper bound for the optimizer-assigned cost (in timerons) of an activity. This value can be any positive big integer (not zero) (SQLSTATE 42820). This condition is enforced at the coordinator member. Activities tracked by this condition are:

- Coordinator activities of type data manipulation language (DML).
- Nested DML activities that are invoked from user logic. Consequently, DML activities that can be initiated by the database manager (such as utilities, procedures, or internal SQL) are not tracked by this condition (unless their cost is included in the parent's estimate, in which case they are indirectly tracked).

SQLROWSRETURNED > *integer-value*

This condition defines an upper bound for the number of rows returned to a client application from the application server. This value can be any positive integer (not zero) (SQLSTATE 42820).This condition is enforced at the coordinator member. Activities tracked by this condition are:

- Coordinator activities of type DML.

- Nested DML activities that are derived from user logic. Activities that are initiated by the database manager through a utility, procedure, or internal SQL are not affected by this condition.

Result sets returned from within a procedure are treated separately as individual activities. There is no aggregation of the rows that are returned by the procedure itself.

ACTIVITYTOTALTIME > *integer-value* **DAY | DAYS | HOUR | HOURS | MINUTE | MINUTES | SECONDS**

This condition defines an upper bound for the amount of time the database manager will allow an activity to execute, including the time the activity was queued. The activities that are covered by this threshold include the execution of SQL statements, not including compilation time, and the load utility. This value can be any positive integer (not zero) (SQLSTATE 42820). Use a valid duration keyword to specify an appropriate unit of time for *integer-value*. This condition is enforced at the coordinator member.

If the specified time unit is SECONDS, the value must be a multiple of 10 (SQLSTATE 42615). The maximum value that can be specified for this threshold is 2 147 483 640 seconds. Any value specified (using the DAY, HOUR, MINUTE, or SECONDS time unit) that has a seconds equivalent larger than 2 147 483 640 seconds will be truncated to this number of seconds.

UOWTOTALTIME > *integer-value* **DAY | DAYS | HOUR | HOURS | MINUTE | MINUTES | SECONDS**

This condition defines an upper bound for the amount of time the database manager will allow a unit of work to execute. This value can be any non-zero positive integer (SQLSTATE 42820). Use a valid duration keyword to specify an appropriate unit of time for *integer-value*. If the specified time unit is SECONDS, the value must be a multiple of 10 (SQLSTATE 42615). This condition is enforced at the coordinator member.

The maximum value that can be specified for this threshold is 2 147 483 640 seconds. If any value (using the DAY, HOUR, MINUTE, or SECONDS time unit) has a seconds equivalent larger than the maximum value, an error is returned (SQLSTATE 42615).

SQLTEMPSPACE > *integer-value* **K | M | G**

This condition defines the maximum amount of system temporary space that can be consumed by an SQL statement on a member. This value can be any positive integer (not zero) (SQLSTATE 42820).

If *integer-value K* (in either upper- or lowercase) is specified, the maximum size is 1024 times *integer-value*. If *integer-value M* is specified, the maximum size is 1 048 576 times *integer-value*. If *integer-value G* is specified, the maximum size is 1 073 741 824 times *integer-value*.

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (subsection execution). Activities that are initiated by the database manager through a utility, procedure, or internal SQL are not affected by this condition.

AGGSQLTEMPSPACE > *integer-value* **K | M | G**

This condition defines the maximum amount of system temporary space that can be consumed by a set of statements in a service class on a member. This value can be any positive integer (not zero) (SQLSTATE 42820).

If *integer-value* K (in either upper- or lowercase) is specified, the maximum size is 1024 times *integer-value*. If *integer-value* M is specified, the maximum size is 1 048 576 times *integer-value*. If *integer-value* G is specified, the maximum size is 1 073 741 824 times *integer-value*.

Activities contributing to the aggregate that is tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work like subsection execution.
- Nested DML activities that are derived from user logic and their corresponding subagent work like subsection execution. Activities initiated by the database manager through a utility, procedure, or internal SQL statement are not affected by this condition.

SQLROWSREAD > *bigint-value*

This condition defines an upper bound on the number of rows that may be read by an activity during its lifetime on a particular member. This value can be any positive big integer (not zero) (SQLSTATE 42820). Note that the number of rows read is different from the number of rows returned, which is controlled by the SQLROWSRETURNED condition.

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.
- Internal SQL activities like those initiated by the setting of a constraint, or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.

CHECKING EVERY *integer-value* **SECOND | SECONDS**

Specifies how frequently the threshold condition is checked for an activity. The threshold is checked at the end of each request (like a fetch operation, for example) and on the interval defined by the CHECKING clause. The CHECKING clause defines an upper bound on how long a threshold violation may go undetected. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820). Setting a low value may impact system performance negatively.

SQLROWSREADINSC > *bigint-value*

This condition defines an upper bound on the number of rows that may be read by an activity on a particular member while it is executing in a service subclass. Rows read before executing in the service subclass specified are not counted. This value can be any positive big integer (not zero) (SQLSTATE 42820). Note that the number of rows read is different from the number of rows returned, which is controlled by the SQLROWSRETURNED condition.

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.
- Internal SQL activities like those initiated by the setting of a constraint, or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.

CHECKING EVERY *integer-value* **SECOND | SECONDS**
Specifies how frequently the threshold condition is checked for an activity. The threshold is checked at the end of each request (like a fetch operation, for example) and on the interval defined by the CHECKING clause. The CHECKING clause defines an upper bound on how long a threshold violation may go undetected. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820). Setting a low value may impact system performance negatively.

CPUTIME > *integer-value* **DAY | DAYS | HOUR | HOURS | MINUTE | MINUTES | SECOND | SECONDS**
This condition defines an upper bound for the amount of processor time that an activity may consume during its lifetime on a particular member. The processor time tracked by this threshold is measured from the time that the activity starts executing. This value can be any positive integer (not zero) (SQLSTATE 42820).

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.
- Internal SQL activities, like those initiated by the setting of a constraint or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.
- Activities of type CALL. For CALL activities, the processor time tracked for the procedure does not include the processor time used by any child activity or by any fenced mode processes. The threshold condition will be checked only upon return from user logic to the database engine. For example: During execution of a trusted routine, the threshold condition will be checked only when the routine issues a request to the database engine.

CHECKING EVERY *integer-value* **SECOND | SECONDS**
Specifies how frequently the threshold condition is checked for an activity. The granularity of the CPUTIME threshold is approximately this number multiplied by the degree of parallelism for the activity. For example: If the threshold is checked every 60

seconds and the degree of parallelism is 2, the activity might use an extra 2 minutes of processor time instead of 1 minute before the threshold violation is detected. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820). Setting a low value may impact system performance negatively.

CPUTIMEINSC > *integer-value* **DAY | DAYS | HOUR | HOURS | MINUTE | MINUTES | SECOND | SECONDS**

This condition defines an upper bound for the amount of processor time that an activity may consume on a particular member while it is executing in a service subclass. The processor time tracked by this threshold is measured from the time that the activity starts executing in the service subclass identified in the threshold domain. Any processor time used before that point is not counted toward the limit imposed by this threshold. This value can be any positive integer (not zero) (SQLSTATE 42820).

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.
- Internal SQL activities, like those initiated by the setting of a constraint or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.
- Activities of type CALL. For CALL activities, the processor time tracked for the procedure does not include the processor time used by any child activity or by any fenced mode processes. The threshold condition will be checked only upon return from user logic to the database engine. For example: During execution of a trusted routine, the threshold condition will be checked only when the routine issues a request to the database engine.

CHECKING EVERY *integer-value* **SECOND | SECONDS**

Specifies how frequently the threshold condition is checked for an activity. The granularity of the CPUTIMEINSC threshold is approximately this number multiplied by the degree of parallelism for the activity. For example: If the threshold is checked every 60 seconds and the degree of parallelism is 2, the activity might use an extra 2 minutes of processor time instead of 1 minute before the threshold violation is detected. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820). Setting a low value may impact system performance negatively.

DATATAGINSC IN *(integer-constant, ...)*

This condition defines one or more data tag values specified on a table space that the activity touches. The data tag on a table space, or its underlying storage group (where applicable), can be either not be set or set to a value from 1 to 9. If the activity touches a table space that has no data tag set (either at the table space or storage group level), this threshold will not have any affect on that activity. The definition domain for this

condition must be a service subclass (SERVICE CLASS specifying the UNDER clause), and the enforcement scope must be DATABASE PARTITION (SQLSTATE 5U037). This condition is enforced independently at each database partition.

Activities tracked by this condition are:

- Coordinator activities of type data manipulation language (DML).
- Nested DML activities that are directly invoked from user logic.

DML activities that can be initiated by the database manager (such as utilities, procedures, or internal SQL) are not tracked by this condition.

This threshold is only checked when a scan is opened on a table or when an insert is performed into a table. Fetching data from a table after a scan has been opened will not violate the threshold.

DATATAGINSC NOT IN *(integer-constant, ...)*
This condition defines one or more data tag values not specified on a table space that the activity touches. The data tag on a table space, or its underlying storage group (where applicable), can be either not be set or set to a value from 1 to 9. If the activity touches a table space that has no data tag set (either at the table space or storage group level), this threshold will not have any affect on that activity. The definition domain for this condition must be a service subclass (SERVICE CLASS specifying the UNDER clause), and the enforcement scope must be DATABASE PARTITION (SQLSTATE 5U037). This condition is enforced independently at each database partition.

Activities tracked by this condition are:

- Coordinator activities of type data manipulation language (DML).
- Nested DML activities that are directly invoked from user logic.

DML activities that can be initiated by the database manager (such as utilities, procedures, or internal SQL) are not tracked by this condition.

This threshold is only checked when a scan is opened on a table or when an insert is performed into a table. Fetching data from a table after a scan has been opened will not violate the threshold.

alter-threshold-exceeded-actions
Specifies what action is to be taken when a condition is exceeded. Each time that a condition is exceeded, an event is recorded in all active threshold violations event monitors.

COLLECT ACTIVITY DATA
Specifies that data about each activity that exceeded the threshold is to be sent to any active activities event monitor when the activity completes. The COLLECT ACTIVITY DATA setting does not apply to non-activity thresholds, such as CONNECTIONIDLETIME, TOTALDBPARTITIONCONNECTIONS, TOTALSCPARTITIONCONNECTIONS, CONCURRENTWORKLOADOCCURRENCES, or UOWTOTALTIME.

alter-collect-activity-data-clause

ON COORDINATOR MEMBER
Specifies that the activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that the activity data is to be collected at all members on

which the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. For predictive thresholds, activity information is collected at all members only if you also specify the CONTINUE action for exceeded thresholds. For reactive thresholds, the ON ALL MEMBERS clause has no effect and activity information is always collected only at the coordinator member. For both predictive and reactive thresholds, any input data values, section information, or values will be collected only at the coordinator member.

WITHOUT DETAILS
Specifies that data about each activity associated with the work class for which this work action is defined should be sent to any active activities event monitor, when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. For predictive thresholds, section actuals will be collected on any member where the activity data is collected. For reactive thresholds, section actuals will be collected only on the coordinator member.

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

NONE
Specifies that activity data should not be collected for each activity that exceeds the threshold.

STOP EXECUTION
The execution of the activity is stopped and an error is returned (SQLSTATE 5U026). In the case of the UOWTOTALTIME threshold, the unit of work is rolled back.

CONTINUE
The execution of the activity is not stopped. When the condition also has a queue, this option causes queuing to extend beyond the size of the queue.

FORCE APPLICATION
The application is forced off the system (SQLSTATE 55032). This action can only be specified for the UOWTOTALTIME threshold.

remap-activity-action

REMAP ACTIVITY TO *service-subclass-name*
The activity is mapped to *service-subclass-name*. The execution of the activity is not stopped. This action is valid only for in-service-class thresholds like

CPUTIMEINSC, SQLROWSREADINSC, DATATAGINSC IN and DATATAGINSC NOT IN thresholds (SQLSTATE 5U037). The *service-subclass-name* must identify an existing service subclass under the same superclass associated with the threshold (SQLSTATE 5U037). The *service-subclass-name* cannot be the same as the associated service subclass of the threshold (SQLSTATE 5U037).

NO EVENT MONITOR RECORD
Specifies that no threshold violation record will be written.

LOG EVENT MONITOR RECORD
Specifies that if a THRESHOLD VIOLATIONS event monitor exists and is active, a threshold violation record is written to it.

ENABLE or DISABLE
Specifies whether or not the threshold is enabled for use by the database manager.

ENABLE
The threshold is used by the database manager to restrict the execution of database activities. Currently running database activities will continue to execute without the restriction of this threshold.

DISABLE
The threshold is not used by the database manager to restrict the execution of database activities. New database activities will not be restricted by this threshold. Thresholds with a queue, for example TOTALSCMEMBERCONNECTIONS or CONCURRENTDBCOORDACTIVITIES, must be disabled before they can be dropped.

Notes

- Thresholds can be defined on different aspects of database behavior to monitor and control that behavior. When a threshold is defined on activities, unless otherwise specified, it will be enforced only during the actual execution of SQL statements, not including compilation time, and the load utility.
- The CONCURRENTWORKLOADOCCURRENCES threshold and the CONCURRENTWORKLOADACTIVITIES threshold differ in scope. CONCURRENTWORKLOADOCCURRENCES controls how many connections can map to a workload definition simultaneously, and CONCURRENTWORKLOADACTIVITIES controls how many activities each connection that is mapped to the workload definition can submit concurrently.
- Changes are written to the system catalog, but do not take effect until after a COMMIT statement, even for the connection that issues the statement.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.
- The new value for a threshold affects only DB2 activities that start executing after the alter operation commits.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

- DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
- TOTALDBPARTITIONCONNECTIONS can be specified in place of TOTALMEMBERCONNECTIONS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
- TOTALSCPARTITIONCONNECTIONS can be specified in place of TOTALSCMEMBERCONNECTIONS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

Example

Alter the threshold MAXBIGQUERIESCONCURRENCY to a maximum of three activities rather than two.

```
ALTER THRESHOLD MAXBIGQUERIESCONCURRENCY
  WHEN CONCURRENTDBCOORDACTIVITIES > 3
  STOP EXECUTION
```

Because this is a threshold with a queue, the threshold cannot be dropped unless it is disabled, as follows:

```
ALTER THRESHOLD MAXBIGQUERIESCONCURRENCY DISABLE
```

ALTER TRIGGER

The ALTER TRIGGER statement changes the description of a trigger at the current server.

Invocation

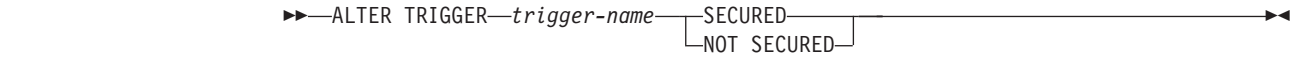
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- SECADM authority
- CREATE_SECURE_OBJECT authority

Syntax



Description

trigger-name
Identifies the trigger to be altered. The *trigger-name* must identify a trigger that exists at the current server (SQLSTATE 42704).

NOT SECURED or SECURED
Specifies whether the trigger is considered secure.

SECURED
Specifies the trigger is considered secure. SECURED must be specified for a trigger whose subject table is a table on which row level or column level access control has been activated (SQLSTATE 428H8). Similarly, SECURED must be specified for a trigger that is created on a view and one or more of the underlying tables in that view definition has row level or column level access control activated (SQLSTATE 428H8).

NOT SECURED
Specifies the trigger is considered not secure. Altering a trigger from secured to not secured fails if the trigger is defined on a table for which row or column level access control is activated (SQLSTATE 428H8). Similarly, altering a trigger from secured to not secured fails if the trigger is defined on a view and one or more of the underlying tables in that view definition has row or column level access control activated (SQLSTATE 428H8).

Examples

- *Example 1*: Alter trigger TRIGGER1 to SECURED.

```
ALTER TRIGGER TRIGGER1 SECURED
```

- *Example 2*: Alter trigger TRIGGER1 to NOT SECURED.

```
ALTER TRIGGER TRIGGER1 NOT SECURED
```

ALTER TRUSTED CONTEXT

The ALTER TRUSTED CONTEXT statement modifies the definition of a trusted context at the current server.

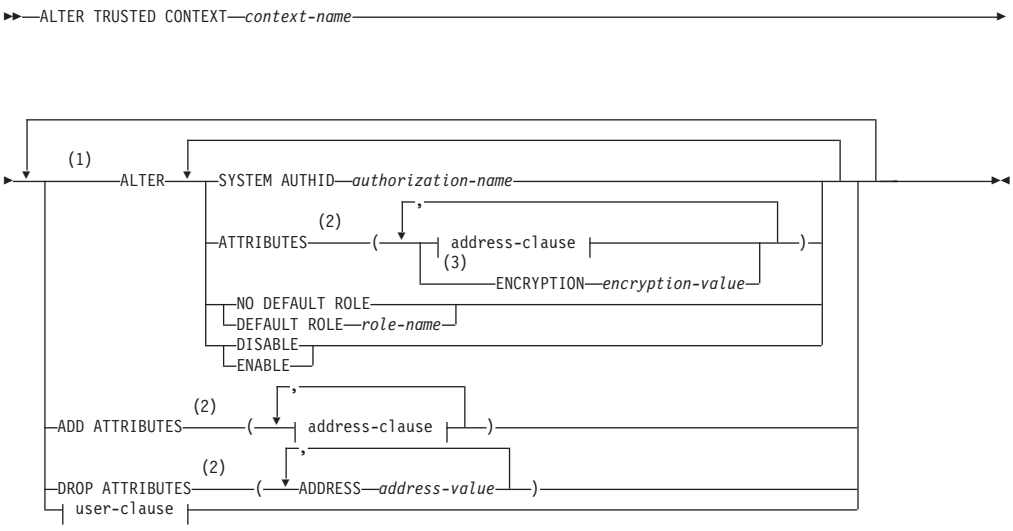
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

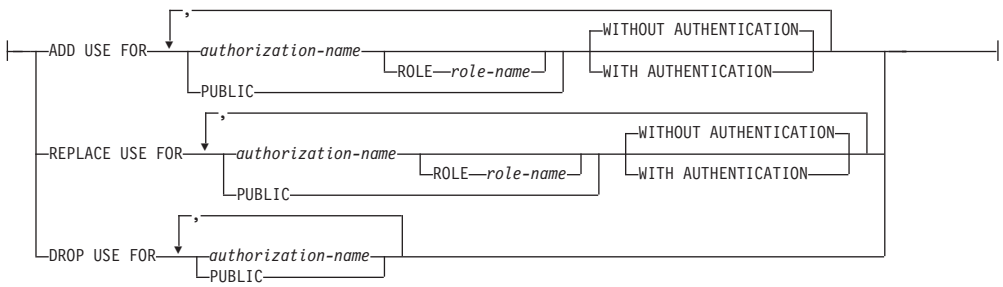
The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



address-clause:

user-clause:



Notes:

1 Each of the ATTRIBUTES, DEFAULT ROLE, ENABLE, and WITH USE clauses can be specified at most once (SQLSTATE 42614).

2 Each attribute name and corresponding value must be unique (SQLSTATE 4274D).

3 ENCRYPTION cannot be specified more than once (SQLSTATE 42614); however, WITH ENCRYPTION can be specified for each ADDRESS that is specified.

Description

context-name
Identifies the trusted context that is to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *context-name* must identify a trusted context that exists at the current server (SQLSTATE 42704).

ALTER
Alters the options and attributes of a trusted context.

SYSTEM AUTHID *authorization-name*
Specifies that the context is a connection established by system authorization ID *authorization-name*, which must not be associated with an existing trusted context (SQLSTATE 428GL). It cannot be the authorization ID of the statement (SQLSTATE 42502).

ATTRIBUTES (...)
Specifies a list of one or more connection trust attributes, upon which the trusted context is defined, that are to be modified. Existing values for the specified attributes are replaced with the new values. If an attribute is not currently part of the trusted context definition, an error is returned (SQLSTATE 4274C). Attributes that are not specified retain their previous values.

ADDRESS *address-value*
Specifies the actual communication address used by the client to communicate with the database server. The only protocol supported is TCP/IP. Previous ADDRESS values for the specified trusted context are removed. The ADDRESS attribute can be specified multiple times, but each *address-value* pair must be unique for the set of attributes (SQLSTATE 4274D).

When establishing a trusted connection, if multiple values are defined for the ADDRESS attribute of a trusted context, a candidate connection is considered to match this attribute if the address used by the connection matches any of the defined values for the ADDRESS attribute of the trusted context.

address-value
Specifies a string constant that contains the value to be associated with the ADDRESS trust attribute. The *address-value* must be an IPv4 address, an IPv6 address, or a secure domain name.

- An IPv4 address must not contain leading spaces and is represented as a dotted decimal address. An example of an IPv4 address is 9.112.46.111. The value 'localhost' or its equivalent representation '127.0.0.1' will not result in a match; the real IPv4 address of the host must be specified instead.

- An IPv6 address must not contain leading spaces and is represented as a colon hexadecimal address. An example of an IPv6 address is 2001:0DB8:0000:0000:0008:0800:200C:417A. IPv4-mapped IPv6 addresses (for example, ::ffff:192.0.2.128) will not result in a match. Similarly, 'localhost' or its IPv6 short representation '::1' will not result in a match.
- A domain name is converted to an IP address by the domain name server where a resulting IPv4 or IPv6 address is determined. An example of a domain name is corona.torolab.ibm.com. When a domain name is converted to an IP address, the result of this conversion could be a set of one or more IP addresses. In this case, an incoming connection is said to match the ADDRESS attribute of a trusted context object if the IP address from which the connection originates matches any of the IP addresses to which the domain name was converted. When creating a trusted context object, it is advantageous to provide domain name values for the ADDRESS attribute instead of static IP addresses, particularly in Dynamic Host Configuration Protocol (DHCP) environments. With DHCP, a device can have a different IP address each time it connects to the network. So, if a static IP address is provided for the ADDRESS attribute of a trusted context object, some device might acquire a trusted connection unintentionally. Providing domain names for the ADDRESS attribute of a trusted context object avoids this problem in DHCP environments.

WITH ENCRYPTION *encryption-value*
Specifies the minimum level of encryption of the data stream or network encryption for this specific *address-value*. This *encryption-value* overrides the global ENCRYPTION attribute setting for this specific *address-value*.

encryption-value
Specifies a string constant that contains the value to be associated with the ENCRYPTION trust attribute for this specific *address-value*. The *encryption-value* must be one of the following values (SQLSTATE 42615):

- NONE, no specific level of encryption is required
- LOW, a minimum of light encryption is required; the authentication type on the database manager must be DATA_ENCRYPT if an incoming connection is to match the encryption setting for this specific address
- HIGH, Secure Sockets Layer (SSL) encryption must be used for data communication between the DB2 client and the DB2 server if an incoming connection is to match the encryption setting for this specific address

ENCRYPTION *encryption-value*
Specifies the minimum level of encryption of the data stream or network encryption. The default is NONE.

encryption-value
Specifies a string constant that contains the value to be associated with the ENCRYPTION trust attribute for this specific *address-value*. The *encryption-value* must be one of the following values (SQLSTATE 42615):

- NONE, no specific level of encryption is required for an incoming connection to match the ENCRYPTION attribute of this trusted context object
- LOW, a minimum of light encryption is required; the authentication type on the database manager must be DATA_ENCRYPT if an incoming connection is to match the ENCRYPTION attribute of this trusted context object
- HIGH, Secure Sockets Layer (SSL) encryption must be used for data communication between the DB2 client and the DB2 server if an incoming connection is to match the ENCRYPTION attribute of this trusted context object

For details about the ENCRYPTION trust attribute, see "CREATE TRUSTED CONTEXT".

NO DEFAULT ROLE or DEFAULT ROLE *role-name*
Specifies whether or not a default role is associated with a trusted connection that is based on this trusted context. If a trusted connection for this context is active, the change comes into effect on the next switch user request or a new connection request.

NO DEFAULT ROLE
Specifies that the trusted context does not have a default role.

DEFAULT ROLE *role-name*
Specifies that *role-name* is the default role for the trusted context. The *role-name* must identify a role that exists at the current server (SQLSTATE 42704). This role is used with the user in a trusted connection, based on this trusted context, when the user does not have a user-specific role defined as part of the definition of the trusted context.

ENABLE or DISABLE
Specifies whether the trusted context is enabled or disabled.

ENABLE
Specifies that the trusted context is enabled.

DISABLE
Specifies that the trusted context is disabled. A trusted context that is disabled is not considered when a trusted connection is established.

ADD ATTRIBUTES
Specifies a list of one or more additional trust attributes on which the trusted context is defined.

ADDRESS *address-value*
Specifies the actual communication address used by the client to communicate with the database server. The only protocol supported is TCP/IP. The ADDRESS attribute can be specified multiple times, but each *address-value* pair must be unique for the set of attributes (SQLSTATE 4274D).

When establishing a trusted connection, if multiple values are defined for the ADDRESS attribute of a trusted context, a candidate connection is considered to match this attribute if the address used by the connection matches any of the defined values for the ADDRESS attribute of the trusted context.

address-value
Specifies a string constant that contains the value to be associated with

the ADDRESS trust attribute. The *address-value* must be an IPv4 address, an IPv6 address, or a secure domain name.

- An IPv4 address must not contain leading spaces and is represented as a dotted decimal address. An example of an IPv4 address is 9.112.46.111. The value 'localhost' or its equivalent representation '127.0.0.1' will not result in a match; the real IPv4 address of the host must be specified instead.
- An IPv6 address must not contain leading spaces and is represented as a colon hexadecimal address. An example of an IPv6 address is 2001:0DB8:0000:0000:0008:0800:200C:417A. IPv4-mapped IPv6 addresses (for example, ::ffff:192.0.2.128) will not result in a match. Similarly, 'localhost' or its IPv6 short representation '::1' will not result in a match.
- A domain name is converted to an IP address by the domain name server, where a resulting IPv4 or IPv6 address is determined. An example of a domain name is corona.torolab.ibm.com.

WITH ENCRYPTION *encryption-value*
Specifies the minimum level of encryption of the data stream or network encryption for this specific *address-value*. This *encryption-value* overrides the global ENCRYPTION attribute setting for this specific *address-value*.

encryption-value
Specifies a string constant that contains the value to be associated with the ENCRYPTION trust attribute for this specific *address-value*. The *encryption-value* must be one of the following values (SQLSTATE 42615):

- NONE, no specific level of encryption is required
- LOW, a minimum of light encryption is required; the authentication type on the database manager must be DATA_ENCRYPT if an incoming connection is to match the encryption setting for this specific address
- HIGH, Secure Sockets Layer (SSL) encryption must be used for data communication between the DB2 client and the DB2 server if an incoming connection is to match the ENCRYPTION attribute of this trusted context object

DROP ATTRIBUTES
Specifies that one or more attributes are to be dropped from the definition of the trusted context. If the attribute and attribute value pair is not currently part of the trusted context definition, an error is returned (SQLSTATE 4274C).

ADDRESS *address-value*
Specifies that the identified communication address is to be removed from the definition of the trusted context. The *address-value* specifies a string constant that contains the value of an existing ADDRESS trust attribute.

ADD USE FOR
Specifies additional users who can use a trusted connection based on this trusted context. If the definition of a trusted context allows access by PUBLIC and a list of users, the specifications for a user override the specifications for PUBLIC.

authorization-name
Specifies that the trusted connection can be used by the specified *authorization-name*. The *authorization-name* must not identify an

authorization ID that is already defined to use the trusted context, and must not be specified more than once in the ADD USE FOR clause (SQLSTATE 428GM). It must also not be the authorization ID of the statement (SQLSTATE 42502).

ROLE *role-name*
Specifies that *role-name* is the role to be used for the user. The *role-name* must identify a role that exists at the current server (SQLSTATE 42704). The role explicitly specified for the user overrides any default role associated with the trusted context.

PUBLIC
Specifies that a trusted connection that is based on this trusted context can be used by any user. PUBLIC must not already be defined to use the trusted context, and PUBLIC must not be specified more than once in the ADD USE FOR clause (SQLSTATE 428GM).

WITHOUT AUTHENTICATION or WITH AUTHENTICATION
Specifies whether or not switching the current user on a trusted connection based on this trusted context requires authentication.

WITHOUT AUTHENTICATION
Specifies that switching the current user on a trusted connection based on this trusted context to this user does not require authentication.

WITH AUTHENTICATION
Specifies that switching the current user on a trusted connection based on this trusted context to this user requires authentication.

REPLACE USE FOR
Specifies that the way in which a particular user or PUBLIC uses the trusted context is to change.

authorization-name
Specifies the *authorization-name* of the user whose use of the trusted connection is to change. The trusted context must already be defined to allow use by the *authorization-name* (SQLSTATE 428GN), and *authorization-name* must not be specified more than once in the REPLACE USE FOR clause (SQLSTATE 428GM). It must also not be the authorization ID of the statement (SQLSTATE 42502).

ROLE *role-name*
Specifies that *role-name* is the role for the user. The *role-name* must identify a role that exists at the current server (SQLSTATE 42704). The role explicitly specified for the user overrides any default role associated with the trusted context.

PUBLIC
Specifies that the attributes for use of the trusted connection by PUBLIC are to change. The trusted context must already be defined to allow use by PUBLIC (SQLSTATE 428GN), and PUBLIC must not be specified more than once in the REPLACE USE FOR clause (SQLSTATE 428GM).

WITHOUT AUTHENTICATION or WITH AUTHENTICATION
Specifies whether or not switching the current user on a trusted connection based on this trusted context requires authentication.

WITHOUT AUTHENTICATION
Specifies that switching the current user on a trusted connection based on this trusted context to this user does not require authentication.

WITH AUTHENTICATION
Specifies that switching the current user on a trusted connection based on this trusted context to this user requires authentication.

DROP USE FOR
Specifies who can no longer use the trusted context. The users who are removed from the definition of the trusted context are those users who are currently allowed to use the trusted context. If one or more, but not all, users can be removed from the definition of the trusted context, the specified users are removed and a warning is returned (SQLSTATE 01682). If none of the specified users can be removed from the definition of the trusted context, an error is returned (SQLSTATE 428GN).

authorization-name
Removes the ability of the specified authorization ID to use this trusted context.

PUBLIC
Removes the ability of all users (except the system authorization ID and individual authorization IDs that have been explicitly enabled) to use this trusted context.

Rules

- A trusted context-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). Trusted context-exclusive SQL statements are:
  - CREATE TRUSTED CONTEXT, ALTER TRUSTED CONTEXT, or DROP (TRUSTED CONTEXT)
- A trusted context-exclusive SQL statement cannot be issued within a global transaction; for example, an XA transaction or a global transaction that is initiated as part of two-phase commit for federated transactions (SQLSTATE 51041).

Notes

- When providing an IP address as part of a trusted context definition, the address must be in the format that is in effect for the network. For example, providing an address in an IPv6 format when the network is IPv4 will not result in a match. In a mixed environment, it is advantageous to specify both the IPv4 and the IPv6 representations of the address, or better yet, to specify a secure domain name (for example, corona.torolab.ibm.com), which hides the address format details.
- Only one uncommitted trusted context-exclusive SQL statement is allowed at a time across all database partitions. If an uncommitted trusted context-exclusive SQL statement is executing, subsequent trusted context-exclusive SQL statements will wait until the current trusted context-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog but do not take effect until they are committed, even for the connection that issues the statement.
- ***Order of operations:*** The order of operations within an ALTER TRUSTED CONTEXT statement is:
  - DROP
  - ALTER
  - ADD ATTRIBUTES
  - ADD USE FOR

  - REPLACE USE FOR
- ***Effect of changes on existing trusted connections:*** If trusted connections exist for the trusted context being altered, the connections remain trusted with the definition in effect before the ALTER TRUSTED CONTEXT statement until the next switch user request or the connection terminates. If the trusted context is disabled while trusted connections for this context are active, the connections remain trusted until the next switch user request or the connection terminates. If trust attributes are changed with the ALTER TRUSTED CONTEXT statement, trusted connections that exist at the time of the ALTER TRUSTED CONTEXT statement that use the trusted context are allowed to continue.
- ***Role privileges:*** If there is no role associated with the user or the trusted context, only the privileges associated with the user are applicable. This is the same as not being in a trusted context.

Examples

- *Example 1:* Assume that trusted context APPSERVER exists and that it is enabled. Issue an ALTER TRUSTED CONTEXT statement to allow Bill to use the trusted context APPSERVER, but put the trusted context in the disabled state.

```
ALTER TRUSTED CONTEXT APPSERVER
  DISABLE
  ADD USE FOR BILL
```

- *Example 2:* Assume that trusted context SECUREROLE exists. Issue an ALTER TRUSTED CONTEXT statement to modify the existing user Joe to use the trusted context with authentication and to add everyone else to use the trusted context without authentication.

```
ALTER TRUSTED CONTEXT SECUREROLE
  REPLACE USE FOR JOE WITH AUTHENTICATION
  ADD USE FOR PUBLIC WITHOUT AUTHENTICATION
```

- *Example 3:* Assume that trusted context SECUREROLEENCRYPT exists with ADDRESS attribute values '9.13.55.100' and '9.12.30.112', and ENCRYPTION attribute value 'NONE'. Issue an ALTER statement to modify the ADDRESS attribute values and the encryption attribute to 'LOW'.

```
ALTER TRUSTED CONTEXT SECUREROLEENCRYPT
  ALTER ATTRIBUTES (ADDRESS '9.12.155.200',
    ENCRYPTION 'LOW')
```

ALTER TYPE (structured)

The ALTER TYPE statement is used to add or drop attributes or method specifications of a user-defined structured type. Properties of existing methods can also be altered.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

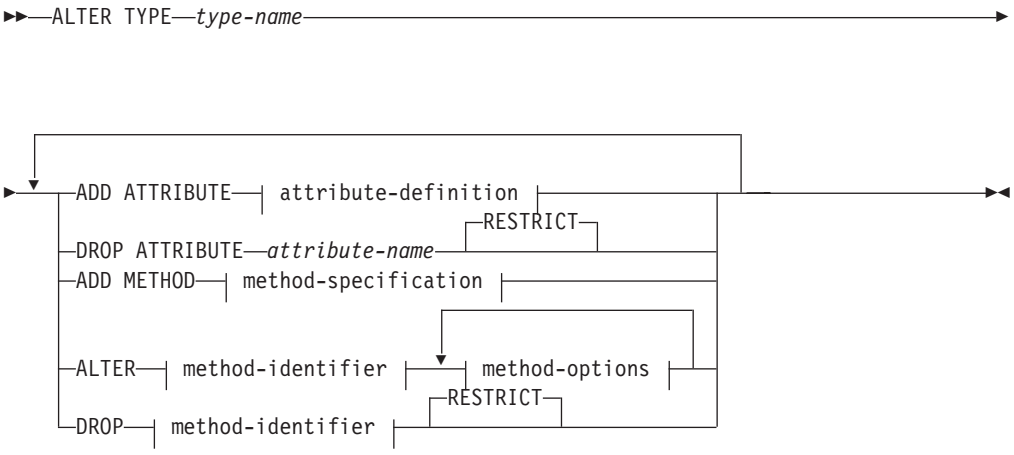
- ALTERIN privilege on the schema of the type
- Owner of the type, as recorded in the OWNER column of the SYSCAT.DATATYPES catalog view
- DBADM authority

To alter a method to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

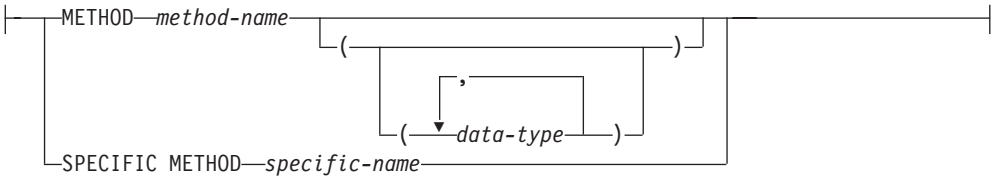
- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

To alter a method to be fenced, no additional authorities or privileges are required.

Syntax



method-identifier:



method-options:

Description

type-name
Identifies the structured type to be changed. It must be an existing type defined in the catalog (SQLSTATE 42704), and the type must be a structured type (SQLSTATE 428DP). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

ADD ATTRIBUTE
Adds an attribute after the last attribute of the existing structured type.

attribute-definition
Defines the attributes of the structured type.

attribute-name
Specifies a name for the attribute. The name cannot be the same as any other attribute of this structured type (including inherited attributes) or any subtype of this structured type (SQLSTATE 42711).

A number of names used as keywords in predicates are reserved for system use, and may not be used as an *attribute-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH and the comparison operators.

data-type 1
Specifies the data type of the attribute. It is one of the data types listed under CREATE TABLE, other than XML (SQLSTATE 42601). The data type must identify an existing data type (SQLSTATE 42704). If *data-type* is specified without a schema name, the type is resolved by searching the schemas on the SQL path. The description of various data types is given in "CREATE TABLE". If the attribute data type is a reference type, the target type of the reference must be a structured type that exists (SQLSTATE 42704).

To prevent type definitions that, at run time, would permit an instance of the type to directly, or indirectly, contain another instance of the same type or one of its subtypes, there is a restriction that a type may not be defined such that one of its attribute types directly or indirectly uses itself (SQLSTATE 428EP).

lob-options
Specifies the options associated with LOB types (or distinct types based on LOB types). For a detailed description of lob-options, see "CREATE TABLE".

DROP ATTRIBUTE
Drops an attribute of the existing structured type.

attribute-name
The name of the attribute. The attribute must exist as an attribute of the type (SQLSTATE 42703).

RESTRICT
Enforces the rule that no attribute can be dropped if *type-name* is used as the type of an existing table, view, column, attribute nested inside the type of a column, or an index extension.

ADD METHOD *method-specification*
Adds a method specification to the type identified by *type-name*. The method cannot be used until a separate CREATE METHOD statement is used to give the method a body. For more information about *method-specification*, see "CREATE TYPE (Structured)".

ALTER *method-identifier*
Uniquely identifies an instance of a method that is to be altered. The specified method may or may not have an existing method body. Methods declared as LANGUAGE SQL cannot be altered (SQLSTATE 42917).

method-identifier

METHOD *method-name*
Identifies a particular method, and is valid only if there is exactly one method instance with the name *method-name* for the type *type-name*. The identified method can have any number of parameters defined for it. If no method by this name exists for the type, an error (SQLSTATE 42704) is raised. If there is more than one instance of the method for the type, an error (SQLSTATE 42725) is raised.

METHOD *method-name (data-type,...)*
Provides the method signature, which uniquely identifies the method. The method resolution algorithm is not used.

method-name
Specifies the name of the method for the type *type-name*.

(data-type,...)
Values must match the data types that were specified (in the corresponding position) on the CREATE TYPE statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific method instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because $0 < n < 25$ means REAL, and $24 < n < 54$ means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the type in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC METHOD *specific-name*
Identifies a particular method, using the name that is specified or defaulted to at method creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific method instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

method-options
Specifies the options that are to be altered for the method.

FENCED or NOT FENCED
Specifies whether the method is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most methods have the option of running as FENCED or NOT FENCED.

If a method is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the method. In general, a method running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for methods that were not adequately coded, reviewed, and tested can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED methods are used.

A method declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a method has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the method cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE methods (SQLSTATE 42849).

THREADSAFE or NOT THREADSAFE
Specifies whether a method is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the method is defined with LANGUAGE other than OLE:

- If the method is defined as THREADSAFE, the database manager can invoke the method in the same process as other routines. In general, to be threadsafe, a method should not use any global or static data areas. Most programming references include a discussion of writing threadsafe

routines. Both FENCED and NOT FENCED methods can be THREADSAFE. If the method is defined with LANGUAGE OLE, THREADSAFE may not be specified (SQLSTATE 42613).

- If the method is defined as NOT THREADSAFE, the database manager will never invoke the method in the same process as another routine. Only a fenced method can be NOT THREADSAFE (SQLSTATE 42613).

DROP *method-identifier*
Uniquely identifies an instance of a method that is to be dropped. The specified method must not have an existing method body (SQLSTATE 428ER). Use the DROP METHOD statement to drop the method body before using ALTER TYPE DROP METHOD. Methods implicitly generated by the CREATE TYPE statement (such as mutators and observers) cannot be dropped (SQLSTATE 42917).

RESTRICT
Indicates that the specified method is restricted from having an existing method body. Use the DROP METHOD statement to drop the method body before using ALTER TYPE DROP METHOD.

Rules

- Adding or dropping an attribute is not allowed for type *type-name* (SQLSTATE 55043) if either:
  - The type or one of its subtypes is the type of an existing table or view.
  - There exists a column of a table whose type directly or indirectly uses *type-name*. The terms *directly uses* and *indirectly uses* are defined in "Structured types".
  - The type or one of its subtypes is used in an index extension.
- A type may not be altered by adding attributes so that the total number of attributes for the type, or any of its subtypes, exceeds 4082 (SQLSTATE 54050).
- ADD ATTRIBUTE option:
  - ADD ATTRIBUTE generates observer and mutator methods for the new attribute. These methods are similar to those generated when a structured type is created (see "CREATE TYPE (Structured)"). If these methods conflict with or override any existing methods or functions, the ALTER TYPE statement fails (SQLSTATE 42745).
  - If the INLINE LENGTH for the type (or any of its subtypes) was explicitly specified by the user with a value less than 292, and the attributes added cause the specified inline length to be less than the size of the result of the constructor function for the altered type (32 bytes plus 10 bytes per attribute), then an error results (SQLSTATE 42611).
- DROP ATTRIBUTE option:
  - An attribute that is inherited from an existing supertype cannot be dropped (SQLSTATE 428DJ).
  - DROP ATTRIBUTE drops the mutator and observer methods of the dropped attributes, and checks dependencies on those dropped methods.
- DROP METHOD option:
  - An original method that is overridden by other methods cannot be dropped (SQLSTATE 42893).

Notes

- It is not possible to alter a method that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

- When a type is altered by adding or dropping an attribute, all packages are invalidated that depend on functions or methods that use this type or a subtype of this type as a parameter or a result.
- When an attribute is added to or dropped from a structured type:
  - If the INLINE LENGTH of the type was calculated by the system when the type was created, the INLINE LENGTH values are automatically modified for the altered type, and all of its subtypes to account for the change. The INLINE LENGTH values are also automatically (recursively) modified for all structured types where the INLINE LENGTH was calculated by the system and the type includes an attribute of any type with a changed INLINE LENGTH.
  - If the INLINE LENGTH of any type affected by adding or dropping attributes was explicitly specified by a user, then the INLINE LENGTH for that particular type is not changed. Special care must be taken for explicitly specified inline lengths. If it is likely that a type will have attributes added later on, then the inline length, for any uses of that type or one of its subtypes in a column definition, should be large enough to account for the possible increase in length of the instantiated object.
  - If new attributes are to be made visible to application programs, existing transform functions must be modified to match the new structure of the data type.
- In a partitioned database environment, the use of SQL in external user-defined functions or methods is not supported (SQLSTATE 42997).
- ***Privileges***: The EXECUTE privilege is not given for any methods explicitly specified in the ALTER TYPE statement until a method body is defined using the CREATE METHOD statement. The owner of the user-defined type has the ability to drop the method specification using the ALTER TYPE statement.

Examples

- *Example 1:* The ALTER TYPE statement can be used to permit a cycle of mutually referencing types and tables. Consider mutually referencing tables named EMPLOYEE and DEPARTMENT.

  The following sequence would allow the types and tables to be created.

```
CREATE TYPE DEPT ...
CREATE TYPE EMP ... (including attribute named DEPTREF of type REF(DEPT))
ALTER TYPE DEPT ADD ATTRIBUTE MANAGER REF(EMP)
CREATE TABLE DEPARTMENT OF DEPT ...
CREATE TABLE EMPLOYEE OF EMP (DEPTREF WITH OPTIONS SCOPE DEPARTMENT)
ALTER TABLE DEPARTMENT ALTER COLUMN MANAGER ADD SCOPE EMPLOYEE
```

  The following sequence would allow these tables and types to be dropped.

```
DROP TABLE EMPLOYEE   (the MANAGER column in DEPARTMENT becomes unscoped)
DROP TABLE DEPARTMENT
ALTER TYPE DEPT DROP ATTRIBUTE MANAGER
DROP TYPE EMP
DROP TYPE DEPT
```

- *Example 2:* The ALTER TYPE statement can be used to create a type with an attribute that references a subtype.

```
CREATE TYPE EMP ...
CREATE TYPE MGR UNDER EMP ...
ALTER TYPE EMP ADD ATTRIBUTE MANAGER REF(MGR)
```

- *Example 3:* The ALTER TYPE statement can be used to add an attribute. The following statement adds the SPECIAL attribute to the EMP type. Because the inline length was not specified on the original CREATE TYPE statement, the DB2 database recalculates the inline length by adding 13 (10 bytes for the new attribute + attribute length + 2 bytes for a non-LOB attribute).

```
ALTER TYPE EMP ...
  ADD ATTRIBUTE SPECIAL CHAR(1)
```

- *Example 4:* The ALTER TYPE statement can be used to add a method associated with a type. The following statement adds a method called BONUS.

```
ALTER TYPE EMP ...
  ADD METHOD BONUS (RATE DOUBLE)
    RETURNS INTEGER
    LANGUAGE SQL
    CONTAINS SQL
    NO EXTERNAL ACTION
    DETERMINISTIC
```

Note that the BONUS method cannot be used until a CREATE METHOD statement is issued to create the method body. If it is assumed that type EMP includes an attribute called SALARY, then the following example shows a method body definition.

```
CREATE METHOD BONUS(RATE DOUBLE) FOR EMP
  RETURN CAST(SELF.SALARY * RATE AS INTEGER)
```

ALTER USAGE LIST

The ALTER USAGE LIST statement alters the definition of a usage list.

Invocation

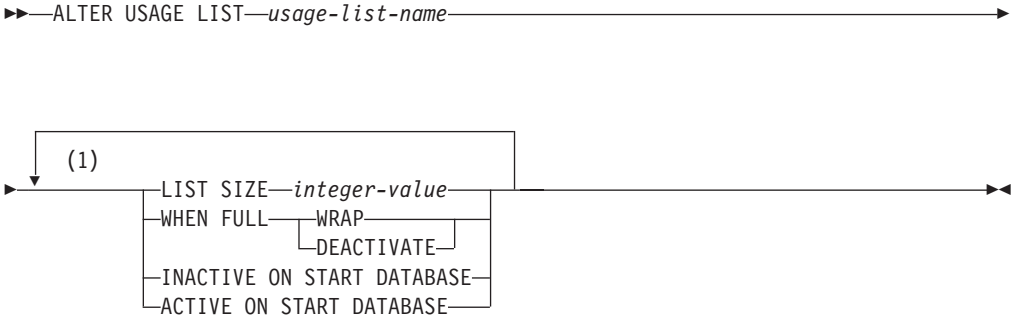
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include one of the following privileges:

- DBADM authority
- SQLADM authority

Syntax



Notes:

1 The same clause cannot be specified more than once

Description

usage-list-name
Identifies the usage list to be altered. The *usage-list-name* must identify a usage list that exists at the current server (SQLSTATE 42704).

LIST SIZE *integer-value*
Specifies that the size of this list is *integer-value* entries. The minimum size that can be specified is 10 and the maximum is 5000 (SQLSTATE 428B7).

WHEN FULL
Specifies the action to perform when an active usage list becomes full.

WRAP
Specifies that the usage list wraps and replaces the oldest entries.

DEACTIVATE
Specifies that the usage list deactivates.

INACTIVE ON START DATABASE
Specifies that the usage list is not activated for monitoring whenever the database is activated. Collection must be explicitly started using the SET USAGE LIST statement.

ACTIVE ON START DATABASE
Specifies that the usage list is automatically activated for monitoring whenever the database is activated. In a partitioned database environment or DB2 pureScale environment, the collection is automatically started whenever the database member is activated.

Notes

- ***When changes take effect***: If the current state of a usage list is active, then the alterations do not take effect when the statement is processed or when the changes are committed. The changes to the usage list take effect the next time the state of usage list is set to active. In a partitioned database environment or DB2 pureScale environment, the alterations take effect the next time the usage list at a member is activated.

ALTER USER MAPPING

The ALTER USER MAPPING statement is used to change the authorization ID or password that is used at a data source for a specified federated server authorization ID.

Invocation

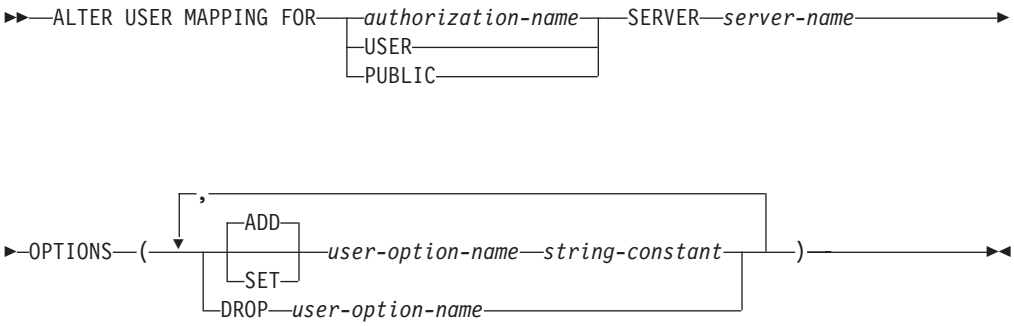
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

If the authorization ID of the statement is different from the authorization name that is mapped to the data source, the privileges held by the authorization ID of the statement must include DBADM authority. Otherwise, if the authorization ID and the authorization name match, no authorities or privileges are required.

When altering a public user mapping, the privileges held by the authorization ID of the statement must include DBADM authority.

Syntax



Description

authorization-name
Specifies the authorization name under which a user or application connects to a federated database.

USER
The value in the special register USER. When USER is specified, then the authorization ID of the ALTER USER MAPPING statement will be mapped to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

PUBLIC
Specifies that any valid authorization ID for the local federated database will be mapped to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

SERVER *server-name*
Identifies the data source accessible under the remote authorization ID that maps to the local authorization ID that's denoted by *authorization-name* or referenced by USER.

OPTIONS
Indicates what user options are to be enabled, reset, or dropped for the mapping that is being altered.

ADD
Enables a user option.

SET
Changes the setting of a user option.

user-option-name
Names a user option that is to be enabled or reset.

string-constant
Specifies the setting for *user-option-name* as a character string constant.

DROP *user-option-name*
Drops a user option.

Notes

- A user option cannot be specified more than once in the same ALTER USER MAPPING statement (SQLSTATE 42853). When a user option is enabled, reset, or dropped, any other user options that are in use are not affected.
- An ALTER USER MAPPING statement within a given unit of work (UOW) cannot be processed (SQLSTATE 55007) if the UOW already includes one of the following items:
  - A SELECT statement that references a nickname for a table or view at the data source that is to be included in the mapping
  - An open cursor on a nickname for a table or view at the data source that is to be included in the mapping
  - Either an INSERT, DELETE, or UPDATE issued against a nickname for a table or view at the data source that is to be included in the mapping.
- Public user mappings and non-public user mappings cannot coexist on the same federated server. This means that if you have created public user mappings, you will not be able to create non-public user mappings on the same federated server. The reverse is also true, if you have created non-public user mappings, you will not be able to create public user mappings on the same federated server.

Examples

- *Example 1:* Jim uses a local database to connect to an Oracle data source called ORACLE1. He accesses the local database under the authorization ID KLEEWEIN; KLEEWEIN maps to CORONA, the authorization ID under which he accesses ORACLE1. Jim is going to start accessing ORACLE1 under a new ID, JIMK. So KLEEWEIN now needs to map to JIMK.

```
ALTER USER MAPPING FOR KLEEWEIN
  SERVER ORACLE1
  OPTIONS ( SET REMOTE_AUTHID 'JIMK'  )
```

- *Example 2:* Mary uses a federated database to connect to a DB2 for z/OS data source called DORADO. She uses one authorization ID to access DB2 and another to access DORADO, and she has created a mapping between these two IDs. She has been using the same password with both IDs, but now decides to use a separate password, ZNYQ, with the ID for DORADO. Accordingly, she needs to map her federated database password to ZNYQ.

```
ALTER USER MAPPING FOR MARY
  SERVER DORADO
  OPTIONS ( ADD REMOTE_PASSWORD 'ZNYQ'  )
```

ALTER VIEW

The ALTER VIEW statement modifies an existing view by altering a reference type column to add a scope. The ALTER VIEW statement also enables or disables a view for use in query optimization.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

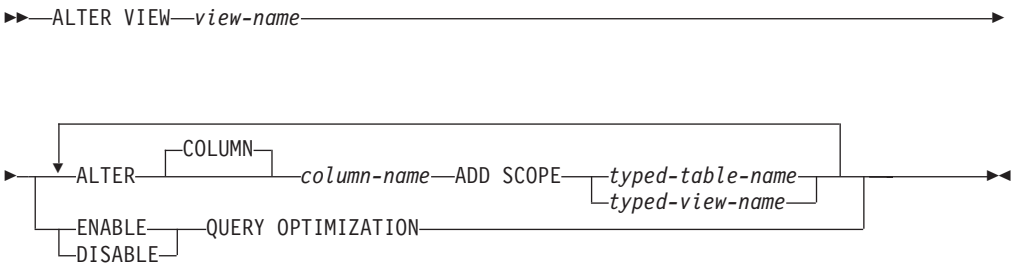
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTERIN privilege on the schema of the view
- Owner of the view to be altered
- CONTROL privilege on the view to be altered
- DBADM authority

To enable or disable a view for use in query optimization, the privileges held by the authorization ID of the statement must also include at least one of the following authorities for each of the tables or underlying tables of views that are referenced in the FROM clause of the view fullselect:

- ALTER privilege on the table
- ALTERIN privilege on the schema of the table
- DBADM authority

Syntax



Description

view-name
Specifies the view that is to be changed. It must be a view that is described in the catalog.

ALTER COLUMN *column-name*
Specifies the name of the column that is to be altered. The *column-name* must identify an existing column of the view (SQLSTATE 42703). The name cannot be qualified.

ADD SCOPE
Adds a scope to an existing reference type column that does not already have a scope defined (SQLSTATE 428DK). The column must not be inherited from a superview (SQLSTATE 428DJ).

typed-table-name
Specifies the name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
Specifies the name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION
Specifies whether or not the view and any associated statistics are to be used to improve the optimization of queries. DISABLE QUERY OPTIMIZATION is the default when a view is created.

ENABLE QUERY OPTIMIZATION
Specifies that the view includes statistics that can be used to improve the optimization of queries that involve this view or queries that include subqueries similar to the fullselect of this view.

DISABLE QUERY OPTIMIZATION
Specifies that the view and any associated statistics are not to be used to improve the optimization of queries.

Rules

- A view cannot be enabled for query optimization if:
  - The view directly or indirectly references a materialized query table (MQT). Note that an MQT or statistical view can reference a statistical view
  - The view directly or indirectly references a catalog table.
  - It is a typed view

Notes

- To be considered for optimizing a query, a view:
  - Cannot contain an aggregation or distinct operation
  - Cannot contain a union, except, or intersect operation
  - Cannot contain an OLAP specification
- If a view is altered to disable query optimization, cached query plans that used the view for query optimization are invalidated. If a view is altered to enable query optimization, cached query plans are invalidated if they reference the same tables as the newly enabled view references, either directly or indirectly through other views. The invalidation of these cached query plans results in implicit revalidation that takes the view's changed query optimization property into account.

  The query optimization property for a view has no impact on static embedded SQL statements.

ALTER WORK ACTION SET

The ALTER WORK ACTION SET statement alters a work action set by adding, altering, or dropping work actions within the work action set.

Invocation

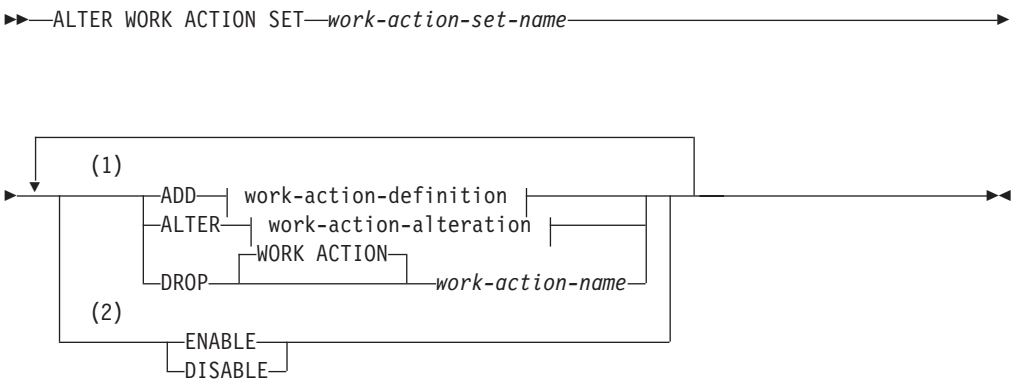
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

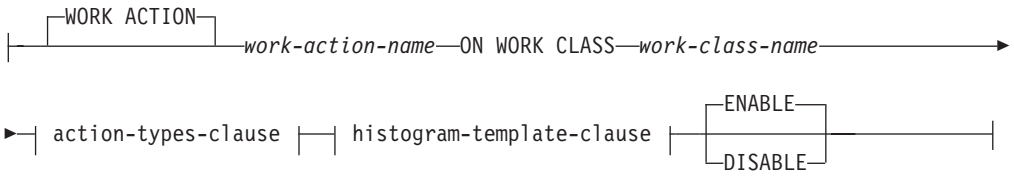
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- SQLADM authority, only if every alteration clause is a COLLECT clause
- WLMADM authority
- DBADM authority

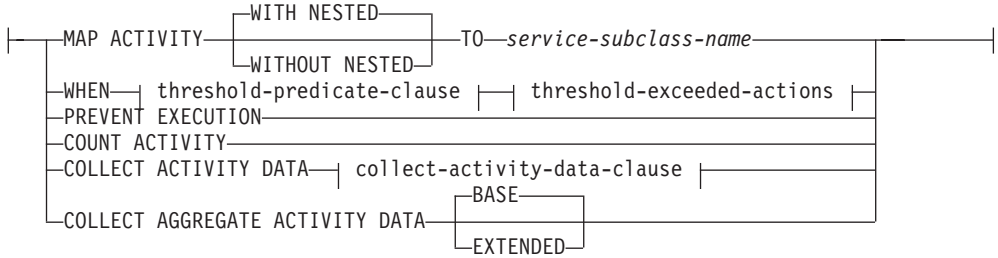
Syntax



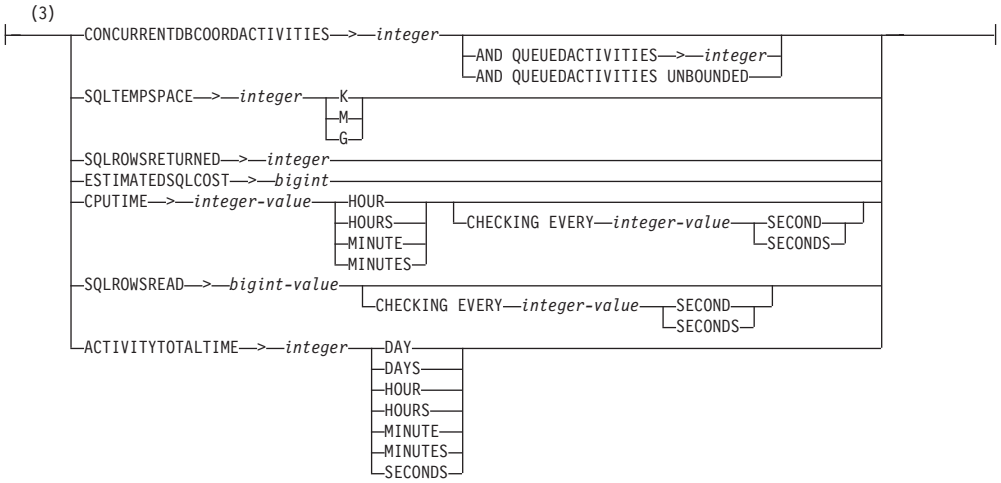
work-action-definition:



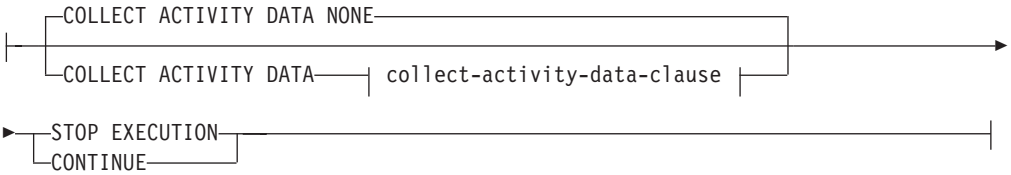
action-types-clause:



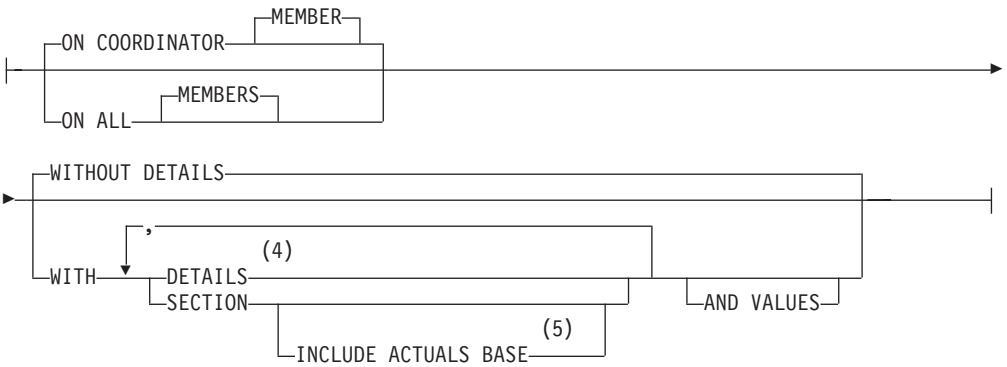
threshold-predicate-clause:



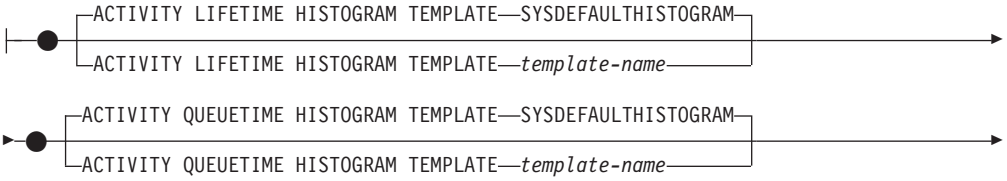
threshold-exceeded-actions:

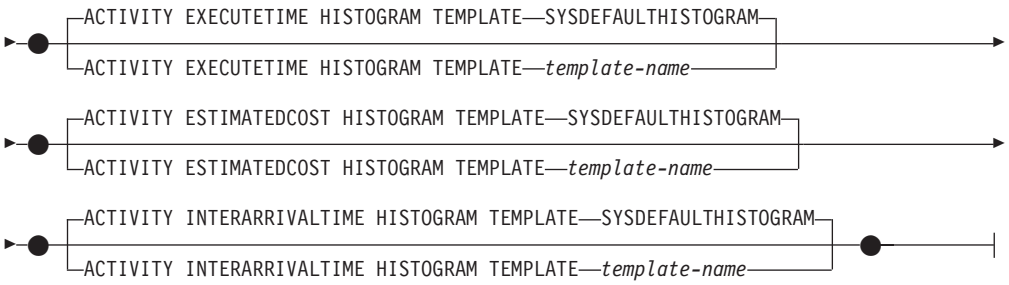


collect-activity-data-clause:

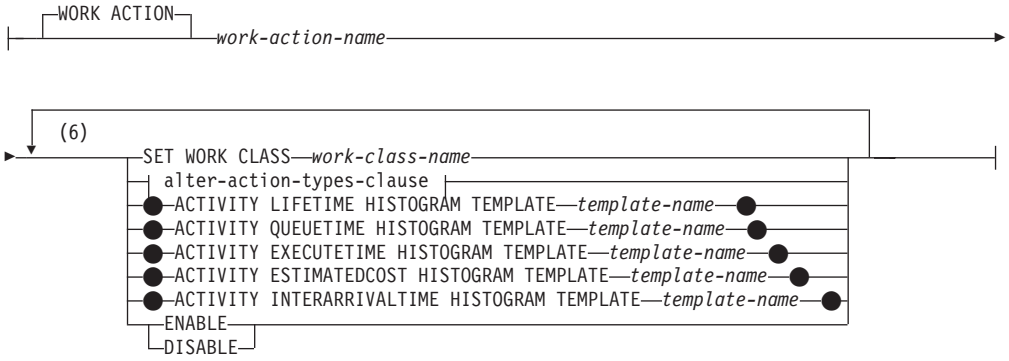


histogram-template-clause:

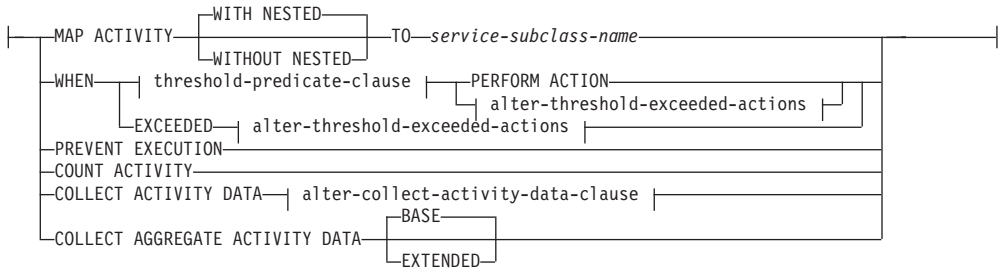




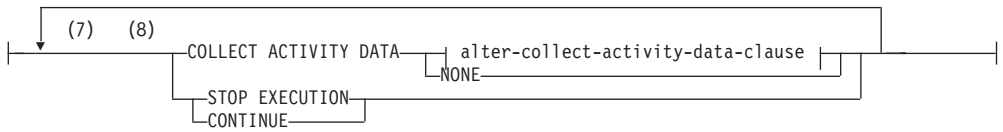
work-action-alteration:



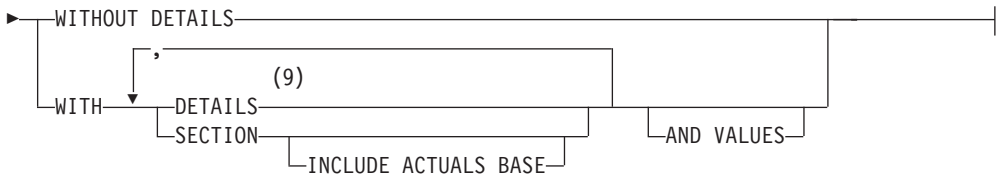
alter-action-types-clause:



alter-threshold-exceeded-actions:



alter-collect-activity-data-clause:



Notes:

1 The ADD, ALTER, and DROP clauses are processed in the order in which they are specified.

2 The ENABLE or DISABLE clause can only be specified once in the same statement.

3 Only one work action of the same threshold type can be applied to a single work class at a time. When altering a threshold work action, the threshold predicate cannot be changed.

4 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

5 This clause does not apply to thresholds.

6 The same clause must not be specified more than once.

7 The same clause must not be specified more than once.

8 If an existing work action does not have a threshold-exceeded action defined for it and it is being altered to become a threshold work action, then either STOP EXECUTION or CONTINUE must be specified, and if COLLECT ACTIVITY DATA is not specified, then COLLECT ACTIVITY DATA NONE is the default.

9 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

Description

work-action-set-name
Identifies the work action set that is to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *work-action-set-name* must identify a work action set that exists at the current server (SQLSTATE 42704).

ADD
Adds a work action to the work action set.

WORK ACTION *work-action-name*
Names the work action. The *work-action-name* must not identify a work action that already exists at the current server under this work action set (SQLSTATE 42710). The *work-action-name* cannot begin with 'SYS' (SQLSTATE 42939).

ON WORK CLASS *work-class-name*
Specifies the work class that identifies the database activities to which this work action will apply. The *work-class-name* must exist in the *work-class-set-name* at the current server (SQLSTATE 42704).

MAP ACTIVITY
Specifies a work action of mapping the activity. This action can only be specified if the object for which this work action set is defined is a service superclass (SQLSTATE 5U034).

WITH NESTED or WITHOUT NESTED
Specifies whether or not activities that are nested under this activity are mapped to the service subclass. The default is WITH NESTED.

WITH NESTED
All database activities that have a nesting level of zero that are classified under the work class, and all database activities nested under this activity, are mapped to the service subclass; that is, activities with a nesting level greater than zero are run under the same service class as activities with a nesting level of zero.

WITHOUT NESTED
Only database activities that have a nesting level of zero that are classified under the work class are mapped to the service subclass. Database activities that are nested under this activity are handled according to their activity type.

TO *service-subclass-name*
Specifies the service subclass to which activities are to be mapped. The *service-subclass-name* must already exist in the *service-superclass-name* at the current server (SQLSTATE 42704). The *service-subclass-name* cannot be the default service subclass, SYSDEFAULTSUBCLASS (SQLSTATE 5U018).

WHEN
Specifies the threshold that will be applied to the database activity that is associated with the work class for which this work action is defined. A threshold can only be specified if the database manager object for which this work action set is defined is a database (SQLSTATE 5U034). None of these thresholds apply to internal database activities initiated by the database manager or to database activities generated by administrative SQL routines.

threshold-predicate-clause
For a description of valid threshold types, see the "CREATE THRESHOLD" statement.

threshold-exceeded-actions
For a description of valid threshold-exceeded actions, see the "CREATE THRESHOLD" statement.

PREVENT EXECUTION
Specifies that none of the database activities associated with the work class for which this work action is defined will be allowed to run (SQLSTATE 5U033).

COUNT ACTIVITY
Specifies that all of the database activities associated with the work class are to be run and that each time one is run, the counter for the work class will be incremented.

COLLECT ACTIVITY DATA
Specifies that data about each activity associated with the work class for which this work action is defined is to be sent to any active activities event monitor when the activity completes.

collect-activity-data-clause

ON COORDINATOR MEMBER
Specifies that the activity data is to be collected at only the coordinator member of the activity.

ON ALL MEMBERS
Specifies that the activity data is to be collected at all members on which the activity is processed. For predictive thresholds, activity information is collected at all members only if you also specify the CONTINUE action for exceeded thresholds. For reactive thresholds, the ON ALL MEMBERS clause has no effect and activity information is always collected only at the coordinator member. For both predictive and reactive thresholds, any input data values, section information, or values will be collected only at the coordinator member.

WITHOUT DETAILS
Specifies that data about each activity associated with the work class for which this work action is defined should be sent to any active activities event monitor, when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment and section environment data is to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified.

INCLUDE ACTUALS BASE
Specifies that section actuals should also be collected on any partition where the activity data is collected. For section actuals to be collected, either INCLUDE ACTUALS clause must be specified or the **section_actuals** database configuration parameter must be set.

The effective setting for the collection of section actuals is the combination of the INCLUDE ACTUALS clause, the **section_actuals** database configuration parameter, and the <collectsectionactuals> setting specified on the WLM_SET_CONN_ENV routine. For example, if INCLUDE ACTUALS BASE is specified, yet the **section_actuals** database configuration parameter value is NONE and <collectsectionactuals> is set to NONE, then the effective setting for the collection of section actuals is BASE.

BASE specifies that the following actuals should be enabled and collected during the activity's execution:

- Basic operator cardinality counts
- Statistics for each object referenced (DML statements only)

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

NONE
Specifies that activity data should not be collected for each activity that is associated with the work class for which this work action is defined.

COLLECT AGGREGATE ACTIVITY DATA
Specifies that aggregate activity data is to be captured for activities that are associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default is COLLECT AGGREGATE ACTIVITY DATA BASE. This clause cannot be specified for a work action defined in a work action set that is applied to a database.

BASE
Specifies that basic aggregate activity data should be captured for activities associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. Basic aggregate activity data includes:

- Estimated activity cost high watermark
- Rows returned high watermark
- Temporary table space usage high watermark. Only activities that have an SQLTEMPSPACE threshold applied to them participate in this high watermark.
- Activity life time histogram
- Activity queue time histogram
- Activity execution time histogram

EXTENDED
Specifies that all aggregate activity data should be captured for activities associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. This includes all basic aggregate activity data plus:

- Activity data manipulation language (DML) estimated cost histogram
- Activity DML inter-arrival time histogram

ENABLE or DISABLE
Specifies whether or not the work action is to be considered when database activities are submitted. The default is ENABLE.

ENABLE
Specifies that the work action is enabled and will be considered when database activities are submitted.

DISABLE
Specifies that the work action is disabled and will not be considered when database activities are submitted.

histogram-template-clause
Specifies histogram templates to use when collecting aggregate activity data for activities associated with the work class to which this work action is assigned. Aggregate activity data is only collected for the work class when the work action type is COLLECT AGGREGATE ACTIVITY DATA.

ACTIVITY LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of DB2

activities—associated with the work class to which this work action is assigned—running during a specific interval. This time includes both time queued and time executing. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY QUEUETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities—associated with the work class to which this work action is assigned—are queued during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities—associated with the work class to which this work action is assigned—are executing during a specific interval. This time does not include the time spent queued. Activity execution time is collected in this histogram at each member where the activity executes. On the activity's coordinator member, this is the end-to-end execution time (that is, the life time less the time spent queued). On non-coordinator members, this is the time that these members spend working on behalf of the activity. During the execution of a given activity, DB2 might present work to a non-coordinator member more than once, and each time the non-coordinator member will collect the execution time for that occurrence of the activity. Therefore, the counts in the execution time histogram might not represent the actual number of unique activities that executed on a member. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY ESTIMATEDCOST HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the estimated cost, in timerons, of DML activities associated with the work class to which this work action is assigned. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

ACTIVITY INTERARRIVALTIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, between the arrival of one DML activity and the arrival of the next DML activity, for any activity associated with the work class to which this work action is assigned. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

ALTER
Alters the definition of the work action. You can change the work class to which this work action applies, and the action that is to be applied to the database activity that falls within the work class.

WORK ACTION *work-action-name*
Identifies the work action. The *work-action-name* must identify a work action that exists at the current server under this work action set (SQLSTATE 42704).

SET WORK CLASS *work-class-name*
Specifies the work class that identifies the database activities to which this work action will apply. The *work-class-name* must exist in the *work-class-set-name* at the current server (SQLSTATE 42704).

MAP ACTIVITY
Specifies a work action of mapping the activity. This action can only be specified if the object for which this work action set is defined is a service superclass (SQLSTATE 5U034).

WITH NESTED or WITHOUT NESTED
Specifies whether or not activities that are nested under this activity are mapped to the service subclass. The default is WITH NESTED.

WITH NESTED
All database activities that have a nesting level of zero that are classified under the work class, and all database activities nested under this activity are mapped to the service subclass.

WITHOUT NESTED
Only database activities that have a nesting level of zero that are classified under the work class are mapped to the service subclass. Database activities that are nested under this activity are handled according to their activity type.

TO *service-subclass-name*
Specifies the service subclass to which activities are to be mapped. The *service-subclass-name* must already exist in the *service-superclass-name* at the current server (SQLSTATE 42704). The *service-subclass-name* cannot be the default service subclass, SYSDEFAULTSUBCLASS (SQLSTATE 5U018).

WHEN
Specifies the threshold to be altered for the database activity that is associated with the work class for which this work action is defined.

threshold-predicate-clause
For a description of valid threshold types, see the "CREATE THRESHOLD" statement.

PERFORM ACTION
When altering the value of the threshold predicate condition, specifies that the threshold exceeded action is not changed. The work action must be a threshold (SQLSTATE 42613).

alter-threshold-exceeded-actions
For a description of valid alter-threshold-exceeded-actions, see threshold-exceeded-actions in the "CREATE THRESHOLD" statement.

EXCEEDED
Specifies to keep the same threshold predicate that was specified originally for this altered threshold. The work action must be a threshold (SQLSTATE 42613).

PREVENT EXECUTION
Specifies that none of the database activities associated with the work class for which this work action is defined will be allowed to run (SQLSTATE 5U033).

COUNT ACTIVITY
Specifies that all of the database activities associated with the work class are to be run and that each time one is run, the counter for the work class will be incremented.

COLLECT ACTIVITY DATA
Specifies that data about each activity associated with the work class for which this work action is defined is to be sent to any active activities event monitor when the activity completes.

alter-collect-activity-data-clause

ON COORDINATOR MEMBER
Specifies that the activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that activity data is to be collected at all members where the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. If the AND VALUES clause is specified, activity input values will be collected only for the members of the coordinator.

WITHOUT DETAILS
Specifies that data about each activity that is associated with the work class for which this work action is defined should be sent to any active activities event monitor when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. Section actuals will be collected on any member where the activity data is collected.

INCLUDE ACTUALS BASE
Specifies that section actuals should also be collected on any partition where the activity data is collected. For section actuals to be collected, either INCLUDE ACTUALS clause must be specified or the **section_actuals** database configuration parameter must be set.

The effective setting for the collection of section actuals is the combination of the INCLUDE ACTUALS clause, the **section_actuals** database configuration parameter, and the

<collectsectionactuals> setting specified on the WLM_SET_CONN_ENV routine. For example, if INCLUDE ACTUALS BASE is specified, yet the **section_actuals** database configuration parameter value is NONE and <collectsectionactuals> is set to NONE, then the effective setting for the collection of section actuals is BASE.

BASE specifies that the following actuals should be enabled and collected during the activity's execution:

- Basic operator cardinality counts
- Statistics for each object referenced (DML statements only)

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

NONE
Specifies that activity data should not be collected for each activity that is associated with the work class for which this work action is defined.

COLLECT AGGREGATE ACTIVITY DATA
Specifies that aggregate activity data is to be captured for activities that are associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default is COLLECT AGGREGATE ACTIVITY DATA BASE. This clause cannot be specified for a work action defined in a work action set that is applied to a database.

BASE
Specifies that basic aggregate activity data should be captured for activities associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. Basic aggregate activity data includes:

- Estimated activity cost high watermark
- Rows returned high watermark
- Temporary table space usage high watermark
- Activity life time histogram
- Activity queue time histogram
- Activity execution time histogram

EXTENDED
Specifies that all aggregate activity data should be captured for activities associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. This includes all basic aggregate activity data plus:

- Activity DML estimated cost histogram
- Activity DML inter-arrival time histogram

ACTIVITY LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of DB2 activities—associated with the work class to which this work action is assigned—running during a specific interval. This time includes both time queued and time executing.

The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY QUEUETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities—associated with the work class to which this work action is assigned—are queued during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities—associated with the work class to which this work action is assigned—are executing during a specific interval. This time does not include the time spent queued. Activity execution time is collected in this histogram at each member where the activity executes. On the activity's coordinator member, this is the end-to-end execution time (that is, the life time less the time spent queued). On non-coordinator members, this is the time that these members spend working on behalf of the activity. During the execution of a given activity, DB2 might present work to a non-coordinator member more than once, and each time the non-coordinator member will collect the execution time for that occurrence of the activity. Therefore, the counts in the execution time histogram might not represent the actual number of unique activities that executed on a member. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY ESTIMATEDCOST HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the estimated cost, in timerons, of data manipulation language (DML) activities associated with the work class to which this work action is assigned. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

ACTIVITY INTERARRIVALTIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, between the arrival of one DML activity and the arrival of the next DML activity, for any activity associated with the work class to which this work action is assigned. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

ENABLE or DISABLE
Specifies whether or not the work action is to be considered when database activities are submitted.

ENABLE
Specifies that the work action is enabled and will be considered when database activities are submitted.

DISABLE
Specifies that the work action is disabled and will not be considered when database activities are submitted.

DROP *work-action-name*
Drops the work action from the work action set. The *work-action-name* must identify a work action that exists at the current server under this work action set (SQLSTATE 42704).

A threshold created as part of a work action set cannot be manipulated directly. You must first disable the work action in order to disable the threshold. You can then drop the work action once the threshold is not being used. For more information, see "Dropping a work action" in the *DB2 Workload Management Guide and Reference*.

ENABLE or DISABLE
Specifies whether or not the work action set is to be considered when database activities are submitted.

ENABLE
Specifies that the work action set is enabled and will be considered when database activities are submitted.

DISABLE
Specifies that the work action set is disabled and will not be considered when database activities are submitted.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (histogram template)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (service class)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (threshold)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (work action set)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (work class set)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (workload)
  - GRANT (workload privileges) or REVOKE (workload privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.
- Thresholds with a queue, for example CONCURRENTDBCOORDACTIVITIES, must be disabled before they can be dropped.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.

- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

Examples

- *Example 1:* Alter the DATABASE_ACTIONS work action set and add two work actions using the work class LARGE_SELECTS. For the work action ONE_CONCURRENT_SELECT, apply a concurrency threshold of 1 to control the number of activities that can run at one time, and allow a maximum of 3 to be queued. For work action BIG_ROWS_RETURNED, limit the number of rows that can be returned by database activities that fall within that class to 1 000 000.

```
ALTER WORK ACTION SET DATABASE_ACTIONS
  ADD WORK ACTION ONE_CONCURRENT_SELECT ON WORK CLASS LARGE_SELECTS
     WHEN CONCURRENTDBCOORDACTIVITIES > 1
        AND QUEUEDACTIVITIES > 3 STOP EXECUTION
  ADD WORK ACTION BIG_ROWS_RETURNED ON WORK CLASS LARGE_SELECTS
     WHEN SQLROWSRETURNED > 1000000 STOP EXECUTION
```

- *Example 2:* Alter the ADMIN_APPS_ACTIONS work action set to alter the MAP_SELECTS work action to map all activities that run in super service class ADMIN_APPS under the work class SELECT_CLASS to the service subclass ALL_SELECTS. Also add a new work action called MAP_UPDATES that maps all activities that would run in the work class UPDATE_CLASS to the service subclass ALL_SELECTS.

```
ALTER WORK ACTION SET ADMIN_APPS_ACTIONS
  ALTER WORK ACTION MAP_SELECTS MAP ACTIVITY TO ALL_SELECTS
  ADD WORK ACTION MAP_UPDATES ON WORK CLASS UPDATE_CLASS
    MAP ACTIVITY TO ALL_SELECTS
```

ALTER WORK CLASS SET

The ALTER WORK CLASS SET statement adds, alters, or drops work classes within a work class set.

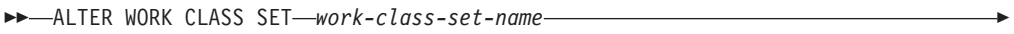
Invocation

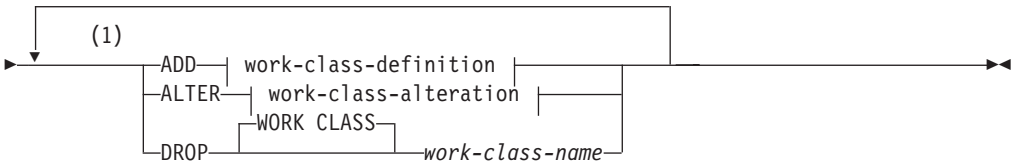
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

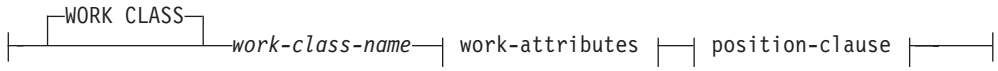
The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

Syntax

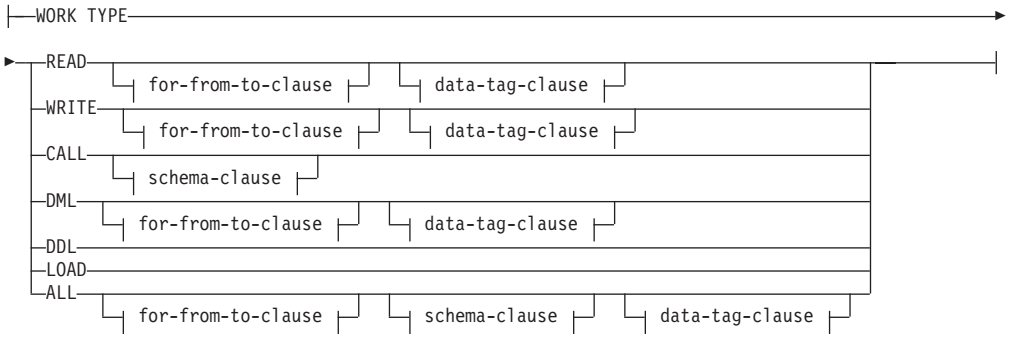




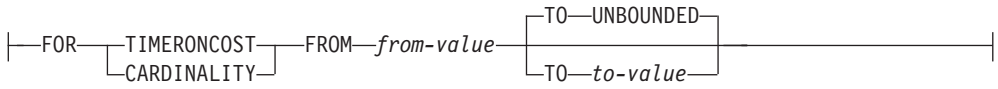
work-class-definition:



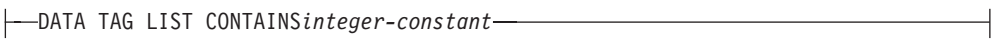
work-attributes:



for-from-to-clause:

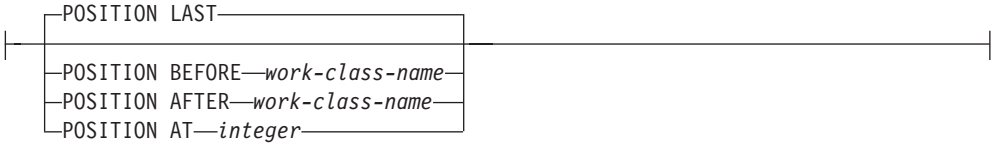


data-tag-clause:

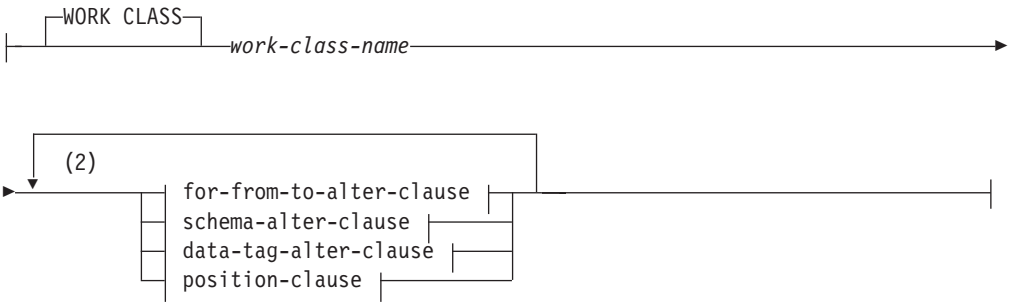


schema-clause:

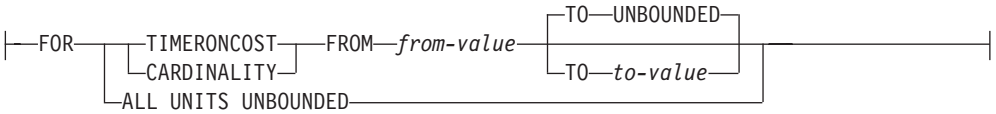
position-clause:



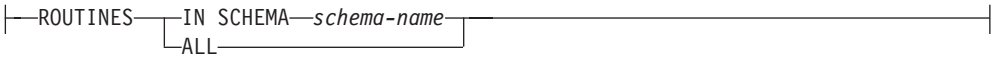
work-class-alteration:



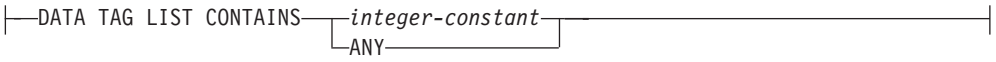
for-from-to-alter-clause:



schema-alter-clause:



data-tag-alter-clause:



Notes:

1 The ADD, ALTER, and DROP clauses are processed in the order in which they are specified.

2 The same clause must not be specified more than once.

Description

work-class-set-name
Identifies the work class set that is to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *work-class-set-name* must identify a work class set that exists at the current server (SQLSTATE 42704).

ADD
Adds a work class to the work class set. For details, see "CREATE WORK CLASS SET".

ALTER
Alters the database activity attributes and the position of a specific work class within the work class set.

WORK CLASS *work-class-name*
Identifies the work class to be altered. The *work-class-name* must identify a work class that exists within the work class set at the current server (SQLSTATE 42704).

DROP
Drops the work class from the work class set.

WORK CLASS *work-class-name*
Identifies the work class to be dropped. The *work-class-name* must identify a work class that exists within the work class set at the current server (SQLSTATE 42704). A work class cannot be dropped if there is a work action in any of the work action sets associated with this work class set that is dependent on it (SQLSTATE 42893).

for-to-from-alter-clause

FOR
Indicates the type of information that is being specified in the FROM *from-value* TO *to-value* clause. The FOR clause is only used for the following work types:

- ALL
- DML
- READ
- WRITE

TIMERONCOST
The estimated cost of the work, in timerons. This value is used to determine whether the work falls within the range specified in the FROM *from-value* TO *to-value* clause.

CARDINALITY
The estimated cardinality of the work. This value is used to determine whether the work falls within the range specified in the FROM *from-value* TO *to-value* clause.

FROM *from-value* **TO UNBOUNDED or FROM** *from-value* **TO** *to-value*
Specifies the range of either timeron value (for estimated cost) or cardinality within which the database activity must fall if it is to be part of this work class. The range is inclusive of *from-value* and *to-value*. This range is only used for the following work types:

- ALL
- DML
- READ

- WRITE

FROM *from-value* **TO UNBOUNDED**
The *from-value* must be zero or a positive DOUBLE value (SQLSTATE 5U019). The range has no upper bound.

FROM *from-value* **TO** *to-value*
The *from-value* must be zero or a positive DOUBLE value and the *to-value* must be a positive DOUBLE value. The *from-value* must be smaller than or equal to the *to-value* (SQLSTATE 5U019).

ALL UNITS UNBOUNDED
Indicates that no range is to be specified in the FROM *from-value* TO *to-value* clause, and that all work that falls within the specified work type is to be included.

schema-alter-clause

ROUTINES
This clause is only used if the work type is CALL or ALL and the database activity is a CALL statement.

IN SCHEMA *schema-name*
Specifies the schema name of the procedure that the CALL statement will be calling.

ALL
Specifies that all schemas are included.

data-tag-alter-clause

DATA TAG LIST CONTAINS *integer-constant*
Specifies the value of the tag given to any data which the database activity might touch if it is to be part of this work class. If the clause is not specified for the work class, all work that falls within the specified work type, regardless of what data it might touch, will be included (that is, the default is to ignore the data tag). This clause is used only if the work type is READ, WRITE, DML, or ALL and the database activity is a DML statement. Valid values for *integer-constant* are integers from 1 to 9.

DATA TAG LIST CONTAINS ANY
Indicates that any data tag setting, including no data tag, is valid for the work class. All work that falls within the specified work type is to be included, regardless of the data tag.

position-clause

POSITION
Specifies where this work class is to be placed within the work class set, which determines the order in which work classes are evaluated. When performing work class assignment at run time, the database manager first determines the work class set that is associated with the object, either the database or a service superclass. The first matching work class within that work class set is then selected. If this keyword is not specified, the work class is placed in the last position.

LAST
Specifies that the work class is to be placed last in the ordered list of work classes within the work class set.

BEFORE *work-class-name*
Specifies that the work class is to be placed before work class

work-class-name in the list. The *work-class-name* must identify a work class in the work class set that exists at the current server (SQLSTATE 42704).

AFTER *work-class-name*
Specifies that the work class is to be placed after work class *work-class-name* in the list. The *work-class-name* must identify a work class in the work class set that exists at the current server (SQLSTATE 42704).

AT *position*
Specifies the absolute position at which the work class is to be placed within the work class set in the ordered list of work classes. This value can be any positive integer (not zero) (SQLSTATE 42615). If *position* is greater than the number of existing work classes plus one, the work class is placed at the last position within the work class set.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (WORK ACTION SET)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (WORK CLASS SET)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.

Examples

- *Example 1:* Alter work class set LARGE_QUERIES and set the two existing work classes to have each range starting at 100 000, keeping the range unbounded. Add a third work class for all SELECT statements that have an estimated timeron cost greater than or equal to 10 000, and position this work class to take priority over the existing two work classes.

```
ALTER WORK CLASS SET LARGE_QUERIES
  ALTER WORK CLASS LARGE_ESTIMATED_COST
   FOR TIMERONCOST FROM 100000 TO UNBOUNDED
  ALTER WORK CLASS LARGE_CARDINALITY
```

```
 FOR CARDINALITY FROM 100000 TO UNBOUNDED
ADD WORK CLASS LARGE_SELECTS WORK TYPE READ
 FOR TIMERONCOST FROM 10000 TO UNBOUNDED POSITION AT 1
```

- *Example 2:* Alter a work class set named DML_STATEMENTS to add a work class that represents all DML SELECT statements that contain a DELETE, INSERT, MERGE, or UPDATE statement.

```
ALTER WORK CLASS SET DML_STATEMENTS
  ADD WORK CLASS UPDATE_CLASS WORK TYPE WRITE
```

ALTER WORKLOAD

The ALTER WORKLOAD statement alters a workload.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

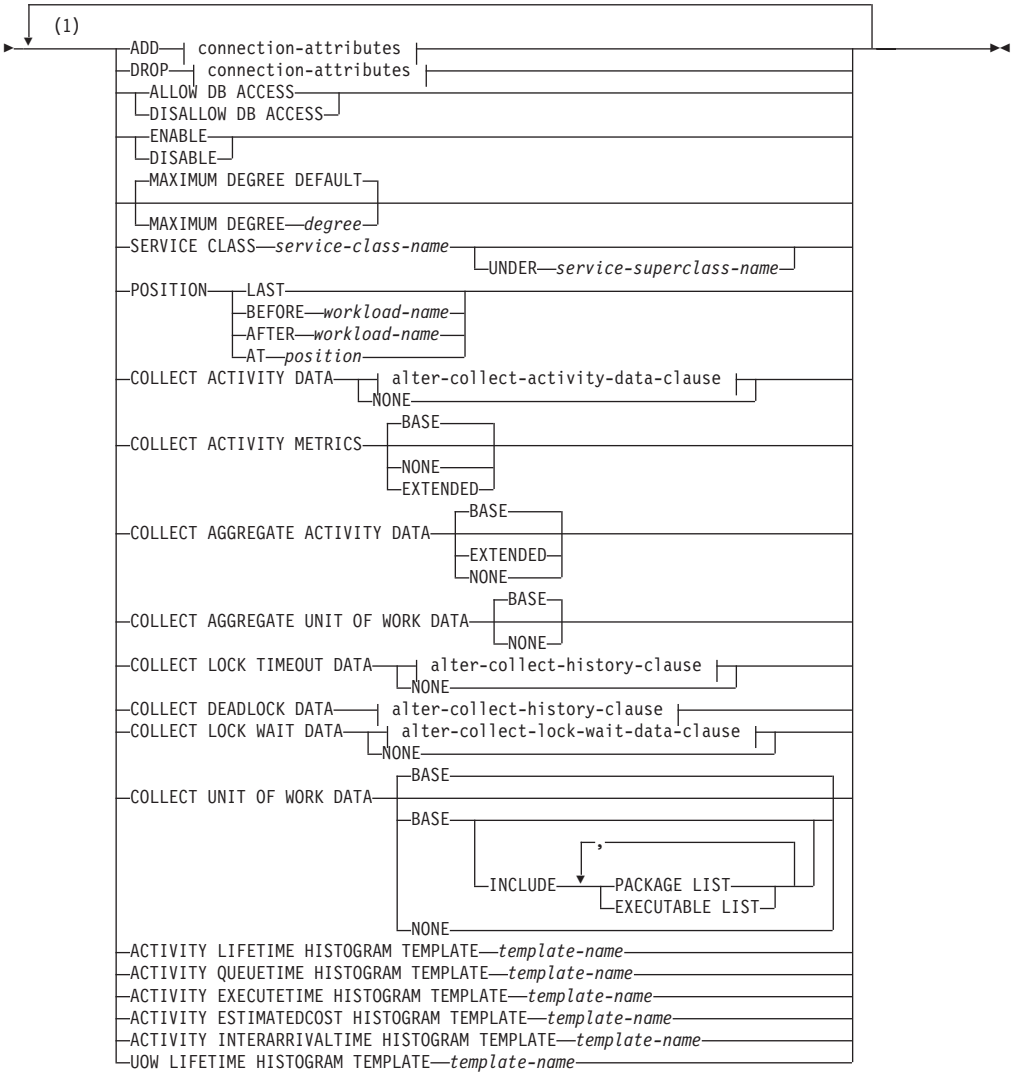
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

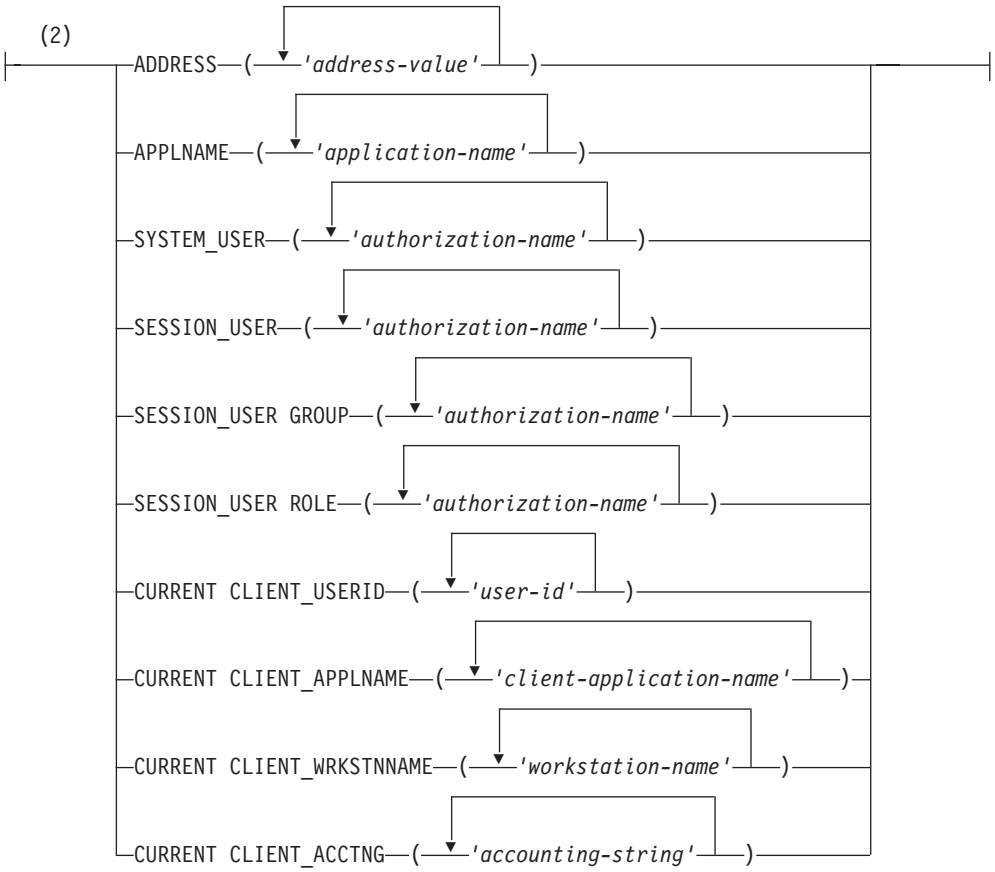
- SQLADM authority, only if every alteration clause is a COLLECT clause
- WLMADM authority
- DBADM authority

To specify any clause other than a COLLECT clause, the authorization ID of the statement must include DBADM or WLMADM authority.

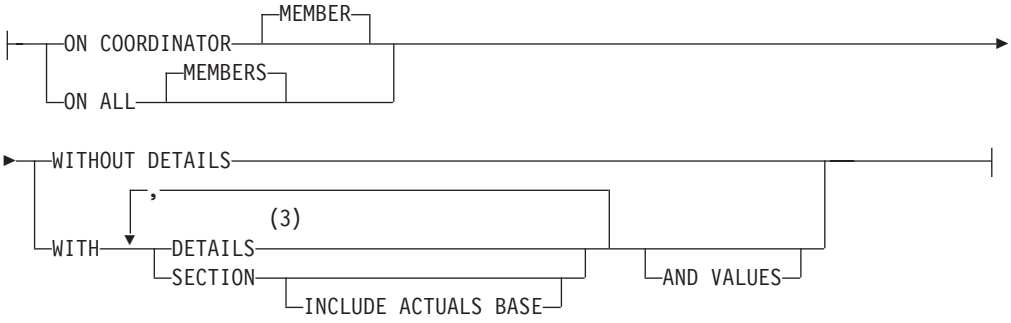
Syntax



connection-attributes:

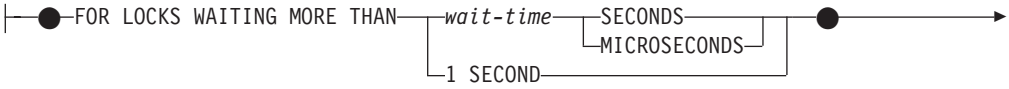


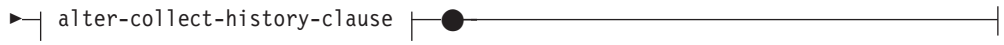
alter-collect-activity-data-clause:



alter-collect-history-clause:

alter-collect-lock-wait-data-clause:





Notes:

1 The same clause must not be specified more than once.

2 Each connection attribute clause can only be specified once.

3 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

Description

workload-name
Identifies the workload that is to be altered. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *workload-name* must identify a workload that exists at the current server (SQLSTATE 42704).

ADD *connection-attributes*
Adds one or more connection attribute values to the definition of the workload. Each specified connection attribute value must not already be defined for the workload (SQLSTATE 5U039). The ADD option cannot be specified if *workload-name* is 'SYSDEFAULTUSERWORKLOAD' or 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

DROP *connection-attributes*
Drops one or more connection attribute values from the definition of the workload. Each specified connection attribute value must be defined for the workload (SQLSTATE 5U040). The DROP option cannot be specified if *workload-name* is 'SYSDEFAULTUSERWORKLOAD' or 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832). There must be at least one defined connection attribute value. The last connection attribute value cannot be dropped (SQLSTATE 5U022).

connection-attributes
Specifies connection attribute values for the workload. All connection attributes are case sensitive, except for ADDRESS.

ADDRESS ('*address-value***', ...)**
Specifies one or more IPv4 addresses, IPv6 addresses or secure domain names for the ADDRESS connection attribute. An address value cannot appear more than once in the list (SQLSTATE 42713). Each address value must be an IPv4 address, an IPv6 address, or a secure domain name.

An IPv4 address must not contain leading spaces and is represented as a dotted decimal address. An example of an IPv4 address is `9.112.46.111`. The value `localhost` or its equivalent representation `127.0.0.1` will not result in a match; the real IPv4 address of the host must be specified instead. An IPv6 address must not contain leading spaces and is represented as a colon hexadecimal address. An example of an IPv6 address is `2001:0DB8:0000:0000:0008:0800:200C:417A`. IPv4-mapped IPv6 addresses (`::ffff:192.0.2.128`, for example) will not result in a match. Similarly, `localhost` or its IPv6 short representation `::1` will not result in a match. A domain name is converted to an IP address by the domain name server where a resulting IPv4 or IPv6 address is determined. An example of a domain name is `corona.torolab.ibm.com`. When a domain name is converted to an IP address, the result of this conversion could be a set of one or more IP addresses. In this case, an incoming connection is said to

match the ADDRESS attribute of a workload object if the IP address from which the connection originates matches any of the IP addresses to which the domain name was converted.

When creating a workload object, you should specify domain name values for the ADDRESS attribute instead of static IP addresses, particularly in Dynamic Host Configuration Protocol (DHCP) environments where a device can have a different IP address each time it connects to the network.

APPLNAME ('*application-name***', ...)**
Specifies one or more applications for the APPLNAME connection attribute. An application name cannot appear more than once in the list (SQLSTATE 42713). If *application-name* does not contain a single asterisk character (*), is equivalent to the value shown in the "Application name" field in system monitor output and in output from the LIST APPLICATIONS command. If *application-name* does contain a single asterisk character (*), the value is used as an expression to represent a set of application names, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the application name, use a sequence of two asterisk characters (**).

SYSTEM_USER ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SYSTEM USER connection attribute. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

SESSION_USER ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SESSION USER connection attribute. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

SESSION_USER GROUP ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SESSION_USER GROUP connection attribute. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

SESSION_USER ROLE ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SESSION_USER ROLE connection attribute. The roles of a session authorization ID in this context refer to all the roles that are available to the session authorization ID, regardless of how the roles were obtained. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

CURRENT CLIENT_USERID ('*user-id***', ...)**
Specifies one or more client user IDs for the CURRENT CLIENT_USERID connection attribute. A client user ID cannot appear more than once in the list (SQLSTATE 42713). If *user-id* contains a single asterisk character (*), the value is used as an expression to represent a set of user IDs, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the user ID, use a sequence of two asterisk characters (**).

CURRENT CLIENT_APPLNAME ('*client-application-name***', ...)**
Specifies one or more applications for the CURRENT CLIENT_APPLNAME connection attribute. An application name cannot appear more than once in the list (SQLSTATE 42713). If *client-application-name* does not contain a single asterisk character (*), is equivalent to the value shown in the "TP Monitor client application name" field in system monitor output. If *client-application-name* does contain a single asterisk character (*), the value is used as an expression to represent

a set of application names, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the application name, use a sequence of two asterisk characters (**).

CURRENT CLIENT_WRKSTNNAME ('*workstation-name***', ...)**
Specifies one or more client workstation names for the CURRENT CLIENT_WRKSTNNAME connection attribute. A client workstation name cannot appear more than once in the list (SQLSTATE 42713). If *workstation-name* contains a single asterisk character (*), the value is used as an expression to represent a set of workstation names, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the workstation name, use a sequence of two asterisk characters (**).

CURRENT CLIENT_ACCTNG ('*accounting-string***', ...)**
Specifies one or more client accounting strings for the CURRENT CLIENT_ACCTNG connection attribute. A client accounting string cannot appear more than once in the list (SQLSTATE 42713). If *accounting-string* contains a single asterisk character (*), the value is used as an expression to represent a set of accounting strings, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the accounting string, use a sequence of two asterisk characters (**).

ALLOW DB ACCESS or DISALLOW DB ACCESS
Specifies whether or not a workload occurrence associated with this workload is allowed access to the database.

ALLOW DB ACCESS
Specifies that workload occurrences associated with this workload are allowed access to the database.

DISALLOW DB ACCESS
Specifies that workload occurrences associated with this workload are not allowed access to the database. The next unit of work associated with this workload will be rejected (SQLSTATE 5U020). Workload occurrences that are already running are allowed to complete. This option cannot be specified if *workload-name* is 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

ENABLE or DISABLE
Specifies whether or not this workload will be considered when a workload is chosen.

ENABLE
Specifies that the workload is enabled and will be considered when a workload is chosen.

DISABLE
Specifies that the workload is disabled and will not be considered when a workload is chosen. This option cannot be specified if *workload-name* is SYSDEFAULTUSERWORKLOAD or SYSDEFAULTADMWORKLOAD (SQLSTATE 42832).

MAXIMUM DEGREE
Specifies the maximum runtime degree of parallelism for this workload. The MAXIMUM DEGREE attribute can not be altered if *workload-name* is SYSDEFAULTADMWORKLOAD.

DEFAULT
Specifies that this workload inherits the intrapartition parallelism setting

from the database manager configuration parameter **intra_parallel**. When **intra_parallel** is set to NO, this workload runs with intrapartition parallelism disabled. When **intra_parallel** is set to YES, this workload runs with intrapartition parallelism enabled. This workload does not specify a maximum runtime degree for assigned applications. Therefore, the actual runtime degree is determined as the lower of the value of **max_querydegree** configuration parameter, the value set by SET RUNTIME DEGREE command, and the SQL statement compilation degree.

degree
Specifies the maximum degree of parallelism for this workload. Valid values are 1 to 32,767. With value 1, the associated requests run with intrapartition parallelism disabled. With value 2 to 32,767, the associated requests run with intrapartition parallelism enabled. The actual runtime degree is determined as the lower of this *degree*, the value of **max_querydegree** configuration parameter, the value set by SET RUNTIME DEGREE command and the SQL statement compilation degree.

SERVICE CLASS *service-class-name*
Specifies that requests associated with this workload are to be executed in the service class *service-class-name*. The *service-class-name* must identify a service class that exists at the current server (SQLSTATE 42704). The *service-class-name* cannot be 'SYSDEFAULTSUBCLASS', 'SYSDEFAULTSYSTEMCLASS', or 'SYSDEFAULTMAINTENANCECLASS' (SQLSTATE 5U032). This option cannot be specified if *workload-name* is 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

UNDER *service-superclass-name*
This clause is used when specifying a service subclass. The *service-superclass-name* identifies the service superclass of *service-class-name*. The *service-superclass-name* must identify a service superclass that exists at the current server (SQLSTATE 42704). The *service-superclass-name* cannot be 'SYSDEFAULTSYSTEMCLASS' or 'SYSDEFAULTMAINTENANCECLASS' (SQLSTATE 5U032).

POSITION
Specifies where this workload is to be placed within the ordered list of workloads. At run time, this list is searched in order for the first workload that matches the required connection attributes. This option cannot be specified if *workload-name* is 'SYSDEFAULTUSERWORKLOAD' or 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

LAST
Specifies that the workload is to be last in the list, before the default workloads SYSDEFAULTUSERWORKLOAD and SYSDEFAULTADMWORKLOAD.

BEFORE *relative-workload-name*
Specifies that the workload is to be placed before workload *relative-workload-name* in the list. The *relative-workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The BEFORE option cannot be specified if *relative-workload-name* is 'SYSDEFAULTUSERWORKLOAD' or 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

AFTER *relative-workload-name*
Specifies that the workload is to be placed after workload *relative-workload-name* in the list. The *relative-workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The AFTER

option cannot be specified if *relative-workload-name* is 'SYSDEFAULTUSERWORKLOAD' or 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

AT *position*
Specifies the absolute position at which the workload is to be placed in the list. This value can be any positive integer (not zero) (SQLSTATE 42615). If *position* is greater than the number of existing workloads plus one, the workload is placed at the last position, just before SYSDEFAULTUSERWORKLOAD and SYSDEFAULTADMWORKLOAD.

COLLECT ACTIVITY DATA
Specifies that data about each activity associated with this workload is to be sent to any active activities event monitor when the activity completes.

alter-collect-activity-data-clause

ON COORDINATOR MEMBER
Specifies that activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that activity data is to be collected at all members where the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. If the AND VALUES clause is specified, activity input values will be collected only for the members of the coordinator.

WITHOUT DETAILS
Specifies that data about each activity that is associated with this workload is to be sent to any active activities event monitor, when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. Section actuals will be collected on any member where the activity data is collected.

INCLUDE ACTUALS BASE
Specifies that section actuals should also be collected on any partition where the activity data is collected. For section actuals to be collected, either INCLUDE ACTUALS clause must be specified or the **section_actuals** database configuration parameter must be set.

The effective setting for the collection of section actuals is the combination of the INCLUDE ACTUALS clause, the **section_actuals** database configuration parameter, and the <collectsectionactuals> setting specified on the WLM_SET_CONN_ENV routine. For example, if INCLUDE

ACTUALS BASE is specified, yet the **section_actuals** database configuration parameter value is NONE and <collectsectionactuals> is set to NONE, then the effective setting for the collection of section actuals is BASE.

BASE specifies that the following actuals should be enabled and collected during the activity's execution:

- Basic operator cardinality counts
- Statistics for each object referenced (DML statements only)

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

NONE
Specifies that activity data is not collected for each activity that is associated with this workload.

COLLECT ACTIVITY METRICS
Specifies that monitor metrics should be collected for an activity submitted by an occurrence of the workload. The default is COLLECT ACTIVITY METRICS NONE.

The effective activity metrics collection setting is the combination of the attribute specified by the COLLECT ACTIVITY METRICS clause on the workload submitting the activity, and the MON_ACT_METRICS database configuration parameter. If either the workload attribute or the configuration parameter has a value other than NONE, metrics will be collected for the activity.

NONE
Specifies that no metrics will be collected for any activity submitted by an occurrence of the workload.

BASE
Specifies that basic metrics will be collected for any activity submitted by an occurrence of the workload.

EXTENDED
Specifies that basic metrics will be collected for any activity submitted by an occurrence of the workload. In addition, specifies that the values for the following monitor elements should be determined with additional granularity:

- **total_section_time**
- **total_section_proc_time**
- **total_routine_user_code_time**
- **total_routine_user_code_proc_time**
- **total_routine_time**

COLLECT AGGREGATE ACTIVITY DATA
Specifies that aggregate activity data about the activities associated with this workload is to be sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default when COLLECT AGGREGATE ACTIVITY DATA is specified is COLLECT AGGREGATE ACTIVITY DATA BASE.

BASE
Specifies that basic aggregate activity data about the activities associated

with this workload is to be sent to the statistics event monitor, if one is active. Basic aggregate activity data includes:

- Estimated activity cost high watermark
- Rows returned high watermark
- Temporary table space usage high watermark. Only activities that have an SQLTEMPSPACE threshold applied to them participate in this high watermark.
- Activity life time histogram
- Activity queue time histogram
- Activity execution time histogram

EXTENDED
Specifies that all aggregate activity data about the activities associated with this workload is to be sent to the statistics event monitor, if one is active. This includes all basic aggregate activity data plus:

- Activity data manipulation language (DML) estimated cost histogram
- Activity DML inter-arrival time histogram

NONE
Specifies that no aggregate activity data is to be collected for this workload.

COLLECT AGGREGATE UNIT OF WORK DATA
Specifies that aggregate unit of work data about the units of work associated with this workload is to be sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default, when COLLECT AGGREGATE UNIT OF WORK DATA is specified, is COLLECT AGGREGATE UNIT OF WORK DATA BASE.

BASE
Specifies that basic aggregate unit of work data about the units of work associated with this workload is to be sent to the statistics event monitor, if one is active. Basic aggregate unit of work data includes:

- Unit of work lifetime histogram

NONE
Specifies that no aggregate unit of work data is to be collected for this workload.

COLLECT LOCK TIMEOUT DATA
Specifies that data about lock timeout events that occur within this workload is sent to any active locking event monitor when the lock event occurs. The lock timeout data is collected on all members. This setting works in conjunction with the **MON_LOCKTIMEOUT** database configuration setting. The setting that produces the most detailed output is honored.

alter-collect-history-clause

WITHOUT HISTORY
Specifies that data about lock events that occur within this workload is sent to any active locking event monitor when the lock event occurs. Past activity history and input values are not sent to the event monitor.

WITH HISTORY
Specifies to collect past activity history in the current unit of work for all of this type of lock events. The activity history buffer will wrap after the maximum size limit is used.

The default limit on the number of past activities to be kept by any one application is 250. If the number of past activities is greater than the limit, only the newest activities are reported. This default value can be overridden using the registry variable DB2_MAX_INACT_STMTS to specify a different value. You can choose a different value for the limit to increase or reduce the amount of system monitor heap used for past activity information.

AND VALUES
Specifies that input data values are to be sent to any active locking event monitor for those activities that have them. These data values will not include LOB data, LONG VARCHAR data, LONG VARGRAPHIC data, structured type data, or XML data. For SQL statements compiled using the REOPT ALWAYS bind option, there will be no REOPT compilation or statement execution data values provided in the event information.

NONE
Specifies that lock timeout data for the workload is not collected at any member.

COLLECT DEADLOCK DATA
Specifies that data about deadlock events that occur within this workload is sent to any active locking event monitor when the lock event occurs. The deadlock data is collected on all members. This setting is only honored if the **MON_DEADLOCK** database configuration parameter is not set to NONE.

alter-collect-history-clause

WITHOUT HISTORY
Specifies that data about lock events that occur within this workload is sent to any active locking event monitor when the lock event occurs. Past activity history and input values are not sent to the event monitor.

WITH HISTORY
Specifies to collect past activity history in the current unit of work for all of these type of lock events. The activity history buffer will wrap after the maximum size limit is used.

The default limit on the number of past activities to be kept by any one application is 250. If the number of past activities is greater than the limit, only the newest activities are reported. This default value can be overridden using the registry variable DB2_MAX_INACT_STMTS to specify a different value. You can choose a different value for the limit to increase or reduce the amount of system monitor heap used for past activity information.

AND VALUES
Specifies that input data values are to be sent to any active locking event monitor for those activities that have them. These data values will not include LOB data, LONG VARCHAR data, LONG VARGRAPHIC data, structured type data, or XML data. For SQL statements compiled using the REOPT ALWAYS bind option, there will be no REOPT compilation or statement execution data values provided in the event information.

COLLECT LOCK WAIT DATA
Specifies that data about lock wait events that occur within this workload is sent to any active locking even monitor when the lock has not been acquired

within *wait-time*. This setting works in conjunction with the **mon_lockwait** and **mon_lw_thresh** database configuration parameters. The setting that produces the most detailed output is honored.

alter-collect-lock-wait-data-clause

FOR LOCKS WAITING MORE THAN *wait-time* **SECONDS | MICROSECONDS) | 1 SECOND**
Specifies that data about lock wait events that occur within this workload is sent to the applicable event monitor when the lock has not been acquired within *wait-time*.

This value can be any non-negative integer. Use a valid duration keyword to specify an appropriate unit of time for *wait-time*. The minimum valid value for the *wait-time* parameter is 1000 microseconds.

WITH HISTORY
Specifies to collect past activity history in the current unit of work for all of this type of lock events. The activity history buffer will wrap after the maximum size limit is used.

The default limit on the number of past activities to be kept by any one application is 250. If the number of past activities is greater than the limit, only the newest activities are reported. This default value can be overridden using the registry variable DB2_MAX_INACT_STMTS to specify a different value. You can choose a different value for the limit to increase or reduce the amount of system monitor heap used for past activity information.

AND VALUES
Specifies that input data values are to be sent to any active locking event monitor for those activities that have them. These data values will not include LOB data, LONG VARCHAR data, LONG VARGRAPHIC data, structured type data, or XML data. For SQL statements compiled using the REOPT ALWAYS bind option, there will be no REOPT compilation or statement execution data values provided in the event information.

NONE
Specifies that the lock wait event for the workload is not collected at any member.

COLLECT UNIT OF WORK DATA
Specifies that data about each unit of work, also referred to as a transaction, associated with this workload is to be sent to the unit of work event monitors, if any have been created, when the unit of work ends. The default is COLLECT UNIT OF WORK BASE. If the **mon_uow_data** database configuration parameter is set to BASE, it takes precedence over the COLLECT UNIT OF WORK DATA parameter. A value of NONE for the **mon_uow_data** indicates that the COLLECT UNIT OF WORK DATA parameters of individual workloads is used.

BASE
Specifies that the base level of data for transactions, associated with this workload, is sent to the unit of work event monitors.

Some of the information reported in a unit of work event are system level request metrics. The collection of these metrics is controlled independently from the collection of the unit of work data. The request metrics are controlled with the COLLECT REQUEST METRICS clause on superclass, or using the **mon_req_metrics** database configuration parameter. The service

super class which the workload is associated with, or the service super class of the service subclass which the workload is associated with, must have the collection of request metrics enabled in order for the request metrics to be present in the unit of work event. If the request metrics collection is not enabled, the value of the request metrics will be zero.

INCLUDE PACKAGE LIST
Specifies that base level of data and the package list for transactions associated with this workload are sent to the unit of work event monitor.

The size of the collected package list is determined by the value of the **mon_pkglist_sz** database configuration parameter. If this value is 0, then the package list is not collected even if the PACKAGE LIST option is specified.

In a partitioned database environment, the package list is only available on the coordinator member. The BASE level will be collected on remote members.

Some of the information reported in a unit of work event are system level request metrics. The collection of these metrics is controlled independently from the collection of the unit of work data. The request metrics are controlled with the COLLECT REQUEST METRICS clause on superclass, or using the **mon_req_metrics** database configuration parameter. The service super class which the workload is associated with, or the service super class of the service subclass which the workload is associated with, must have the collection of request metrics enabled in order for the request metrics to be present in the unit of work event. If the request metrics collection is not enabled, the value of the request metrics will be zero.

INCLUDE EXECUTABLE LIST
Specifies that executable ID list will be collected for a unit of work together with base level of data and sent to the unit of work event monitor.

NONE
Specifies that no unit of work data for transactions associated with this workload is sent to the unit of work event monitor.

ACTIVITY LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of DB2 activities running in the workload during a specific interval. This time includes both time queued and time executing. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY QUEUETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the workload are queued during a specific interval. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the workload are executing during a specific interval. This time does not include the time spent queued. Activity execution time is collected in this histogram at the coordinator member only. The time does not include idle time. Idle time is the time between the execution of requests belonging to the

same activity when no work is being done. An example of idle time is the time between the end of opening a cursor and the start of fetching from that cursor. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY ESTIMATEDCOST HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the estimated cost, in timerons, of DML activities running in the workload. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

ACTIVITY INTERARRIVALTIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, between the arrival of one DML activity into this workload and the arrival of the next DML activity into this workload. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

UOW LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of units of work running in the workload during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE UNIT OF WORK DATA clause is specified with the BASE option.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (WORK ACTION SET)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (WORK CLASS SET)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement. For newly submitted workload occurrences, changes take effect after the ALTER WORKLOAD statement commits. For active workload occurrences, changes take effect at the beginning of the next unit of work.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is

executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.

- If the DISABLE option is specified, the workload is disabled after the statement commits. The workload is not considered the next time that a workload is chosen. If there is an active workload occurrence associated with this workload when the ALTER WORKLOAD statement commits, it continues to run until the end of the current unit of work. At the beginning of the next unit of work, a workload re-evaluation takes place, and the connection becomes associated with a different workload.
- ***Privileges***: The USAGE privilege is not granted to any user, group, or role when a workload is created. To enable use of a workload, grant USAGE privilege on that workload to a user, a group, or a role using the GRANT USAGE ON WORKLOAD statement.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - COLLECT UNIT OF WORK DATA PACKAGE LIST can be specified in place of COLLECT UNIT OF WORK DATA BASE INCLUDE PACKAGE LIST.

Examples

- *Example 1:* The workload PAYROLL is currently positioned such that the workload INVENTORY is considered first when DB2 chooses a workload at run time. Alter the evaluation order so that PAYROLL will be considered first.

```
ALTER WORKLOAD PAYROLL
  POSITION BEFORE INVENTORY
```

- *Example 2:* Alter the evaluation order so that the workload BENCHMARK is evaluated by DB2 before any other workload in the catalog.

```
ALTER WORKLOAD BENCHMARK
  POSITION AT 1
```

- *Example 3:* The workload REPORTS was created with APPLNAME set to appl1, appl2, and appl3, and SYSTEM_USER set to BOB and MARY. Alter the workload to add a new application, appl4 to the application name list, and remove appl2, because it should no longer be mapped to REPORTS.

```
ALTER WORKLOAD REPORTS
  ADD APPLNAME ('appl4')
  DROP APPLNAME ('appl2')
```

- *Example 4:* Assuming a lock event monitor called LOCK exists and is active, create lock event records with statement history for lock timeout events that occur within the workload APP.

```
ALTER WORKLOAD APP
  COLLECT LOCK TIMEOUT DATA WITH HISTORY
```

- *Example 5:* Assuming a lock event monitor called LOCK exists and is active, create lock event records for only deadlock and lock timeout events that occur within the workload PAYROLL on all partitions.

```
ALTER WORKLOAD PAYROLL
  COLLECT DEADLOCK DATA
  COLLECT LOCK TIMEOUT DATA WITHOUT HISTORY
```

- *Example 6:* Assuming a lock event monitor called LOCK exists and is active, create lock event records with statement history and values for deadlock events that occur within the workload INVOICE.

```
ALTER WORKLOAD INVOICE
  COLLECT DEADLOCK DATA WITH HISTORY AND VALUES
```

- *Example 7:* Assuming a lock event monitor called LOCK exists and is active, create lock event records with statement history and values for locks acquired after waiting for more than 150 milliseconds that occur within the workload INVOICE.

```
ALTER WORKLOAD INVOICE
  COLLECT LOCK WAIT DATA FOR LOCKS WAITING MORE THAN 150000
  MICROSECONDS WITH HISTORY AND VALUES
```

- *Example 8:* Alter the workload REPORTS to collect unit of work data and send it to the unit of work event monitor:

```
ALTER WORKLOAD REPORTS
  COLLECT UNIT OF WORK DATA BASE
```

ALTER WRAPPER

The ALTER WRAPPER statement is used to update the properties of a wrapper.

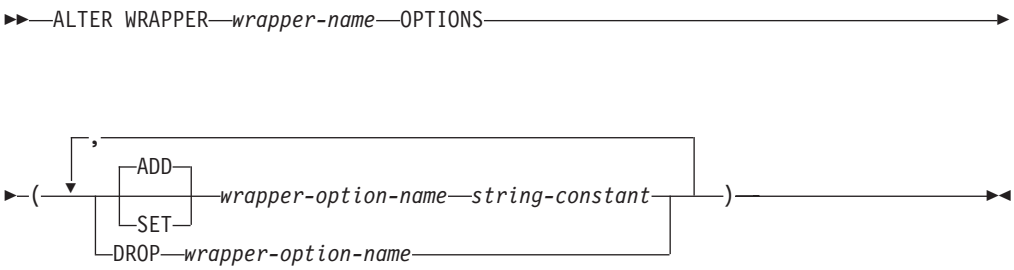
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include DBADM authority.

Syntax



Description

wrapper-name
Specifies the name of the wrapper.

OPTIONS
Indicates what wrapper options are to be enabled, reset, or dropped.

ADD
Enables a server option.

SET
Changes the setting of a wrapper option.

wrapper-option-name
Names a wrapper option that is to be enabled or reset. Currently the only supported wrapper option name is DB2_FENCED.

string-constant
Specifies the setting for *wrapper-option-name* as a character string constant. Valid values are 'Y' or 'N'. The default value for relational wrappers is 'N', and the default value for non-relational wrappers is 'Y'.

DROP *wrapper-option-name*
Drops a wrapper option.

Notes

- Execution of the ALTER WRAPPER statement does not include checking the validity of wrapper-specific options.

- An ALTER WRAPPER statement within a given unit of work (UOW) cannot be processed (SQLSTATE 55007) if the UOW already includes one of the following items:
  - A SELECT statement that references a nickname that belongs to the wrapper.
  - An open cursor on a nickname that belongs to the wrapper.
  - An INSERT, DELETE, or UPDATE statement issued against a nickname that belongs to the wrapper.

Example

Set the DB2_FENCED option on for wrapper NET8.

```
ALTER WRAPPER NET8 OPTIONS (SET DB2_FENCED 'Y')
```

ALTER XSROBJECT

This statement is used to either enable or disable the decomposition support for a specific XML schema. Annotated XML schemas can be used to decompose XML documents into relational tables, if decomposition has been enabled for those XML schemas.

Invocation

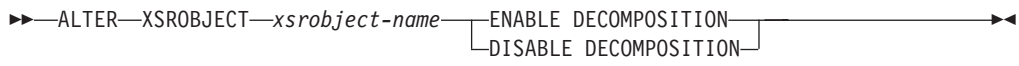
The ALTER XSROBJECT statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if the DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

One of the following authorities is required:

- DBADM
- ALTERIN on the SQL schema
- Ownership of the XSR object to be altered

Syntax



Description

xsrobject-name
Identifies the XSR object to be altered. The *xsrobject-name*, including the implicit or explicit schema qualifier, must uniquely identify an existing XSR object at the current server. If no XSR object with this identifier exists, an error is returned (SQLSTATE 42704).

ENABLE DECOMPOSITION or DISABLE DECOMPOSITION
Enables or disables the use of the XSR object for decomposition. The identified XSR object must be an XML schema (SQLSTATE 42809). In order to enable decomposition, the XML schema needs to be annotated with decomposition rules (SQLSTATE 225DE) and the objects referenced by the decomposition rules must exist at the current server (SQLSTATE 42704).

Notes

- When decomposition for an XSR object is disabled, all related catalog entries are removed.
- Decomposition support for an XSR object will be disabled if any objects the XSR object depends on (such as tables) are dropped or altered to become incompatible with the XSR object.
- In a partitioned database environment, you can issue this statement by connecting to any partition.

ASSOCIATE LOCATORS

The ASSOCIATE LOCATORS statement gets the result set locator value for each result set returned by a procedure.

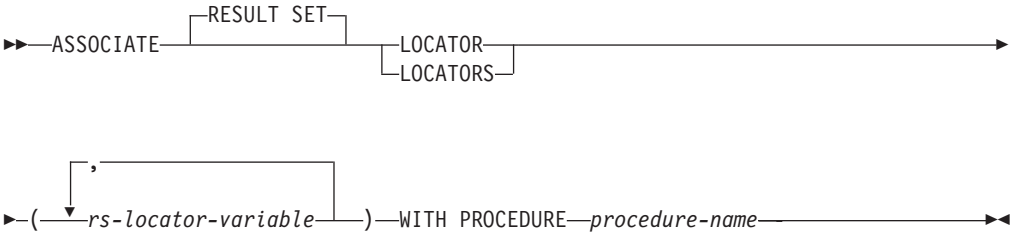
Invocation

This statement can only be embedded in an SQL procedure. It is not an executable statement and cannot be dynamically prepared.

Authorization

None required.

Syntax



Description

rs-locator-variable
Specifies a result set locator variable that has been declared in a compound SQL (Procedure) statement.

WITH PROCEDURE
Identifies the procedure that returns result set locators by the specified procedure name.

procedure-name
A procedure name is a qualified or unqualified name.

A fully qualified procedure name is a two-part name. The first part is an identifier that contains the schema name of the procedure. The last part is an identifier that contains the name of the procedure. A period must separate each of the parts. Any or all of the parts can be a delimited identifier.

If the procedure name is unqualified, it has only one name because the implicit schema name is not added as a qualifier to the procedure name. Successful execution of the ASSOCIATE LOCATOR statement only requires that the unqualified procedure name in the statement be the same as the procedure name in the most recently executed CALL statement that was specified with an unqualified procedure name. The implicit schema name for the unqualified name in the CALL statement is not considered in the match. The rules for how the procedure name must be specified are described in the following paragraph.

When the ASSOCIATE LOCATORS statement is executed, the procedure name or specification must identify a procedure that the requester has already invoked using the CALL statement. The procedure name in the ASSOCIATE LOCATORS statement must be specified the same way that it was specified on

the CALL statement. For example, if a two-part name was specified on the CALL statement, you must use a two-part name in the ASSOCIATE LOCATORS statement.

Notes

- If the number of result set locator variables that are listed in the ASSOCIATE LOCATORS statement is less than the number of locators returned by the procedure, all variables in the statement are assigned a value, and a warning is issued.
- If the number of result set locator variables that are listed in the ASSOCIATE LOCATORS statement is greater than the number of locators returned by the procedure, the extra variables are assigned a value of 0.
- If a procedure is called more than once from the same caller, only the most recent result sets are accessible.
- Result set locator values are available for a procedure that is called using an EXECUTE statement executing the CALL statement that was previously prepared by the PREPARE statement. Result set locator values, however, are not available for a procedure that is called using an EXECUTE IMMEDIATE statement.
- Module-procedure names referenced in an ASSOCIATE LOCATORS statement can only be 1-part or 2-part qualified name references. A 3-part name reference is not allowed (SQLSTATE 42601). Any CALL statement that references a module-procedure that was referenced in an ASSOCIATE LOCATORS statement, must specify the module-procedure with the same 1-part or 2-part qualified name used in the ASSOCIATE LOCATORS statement.

Examples

The statements in the following examples are assumed to be embedded in SQL Procedures.

- *Example 1*: Use result set locator variables LOC1 and LOC2 to get the result set locator values for the two result sets returned by procedure P1. Assume that the procedure is called with a one-part name.

```
CALL P1;
ASSOCIATE RESULT SET LOCATORS (LOC1, LOC2)
  WITH PROCEDURE P1;
```

- *Example 2*: Repeat the scenario in Example 1, but use a two-part name to specify an explicit schema name for the procedure to ensure that procedure P1 in schema MYSCHEMA is used.

```
CALL MYSCHEMA.P1;
ASSOCIATE RESULT SET LOCATORS (LOC1, LOC2)
  WITH PROCEDURE MYSCHEMA.P1;
```

AUDIT

The AUDIT statement determines the audit policy that is to be used for a particular database or database object at the current server. Whenever the object is in use, it is audited according to that policy.

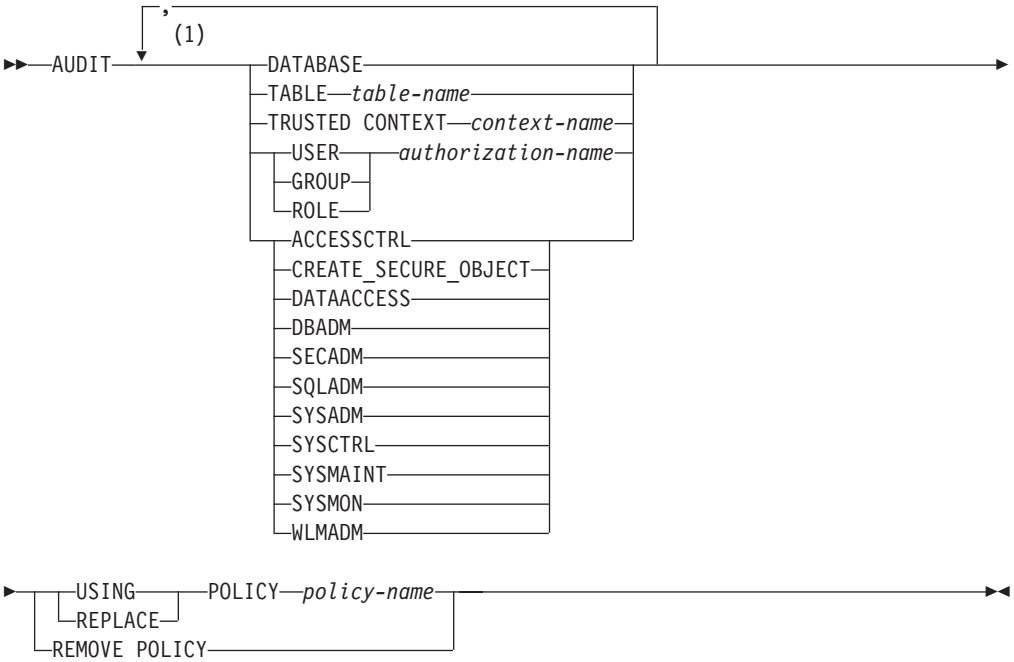
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Notes:

1 Each clause (with the same object name, if applicable) can be specified at most once (SQLSTATE 42713).

Description

ACCESSCTRL, CREATE_SECURE_OBJECT, DATAACCESS, DBADM, SECADM, SQLADM, SYSADM, SYSCTRL, SYSMAINT, SYSMON, or WLMADM

Specifies that an audit policy is to be associated with or removed from the specified authority. All auditable events that are initiated by a user who holds the specified authority, even if that authority is not required for the event, will be audited according to the associated audit policy.

DATABASE
Specifies that an audit policy is to be associated with or removed from the database at the current server. All auditable events that occur within the database are audited according to the associated audit policy.

TABLE *table-name*
Specifies that an audit policy is to be associated with or removed from *table-name*. The *table-name* must identify a table, materialized query table (MQT), or nickname that exists at the current server (SQLSTATE 42704). It cannot be a view, a catalog table, a created temporary table, a declared temporary table (SQLSTATE 42995), or a typed table (SQLSTATE 42997). Only EXECUTE category audit events, with or without data, will be generated when the table is accessed, even if the policy indicates that other categories should be audited.

TRUSTED CONTEXT *context-name*
Specifies that an audit policy is to be associated with or removed from *context-name*. The *context-name* must identify a trusted context that exists at the current server (SQLSTATE 42704). All auditable events that happen within the trusted connection defined by the trusted context *context-name* will be audited according to the associated audit policy.

USER *authorization-name*
Specifies that an audit policy is to be associated with or removed from the user with authorization ID *authorization-name*. All auditable events that are initiated by *authorization-name* will be audited according to the associated audit policy.

GROUP *authorization-name*
Specifies that an audit policy is to be associated with or removed from the group with authorization ID *authorization-name*. All auditable events that are initiated by users who are members of *authorization-name* will be audited according to the associated audit policy. If user membership in a group cannot be determined, the policy will not apply to that user.

ROLE *authorization-name*
Specifies that an audit policy is to be associated with or removed from the role with authorization ID *authorization-name*. The *authorization-name* must identify a role that exists at the current server (SQLSTATE 42704). All auditable events that are initiated by users who are members of *authorization-name* will be audited according to the associated audit policy. Indirect role membership through other roles or groups is valid.

USING, REMOVE, or REPLACE
Specifies whether the audit policy should be used, removed, or replaced for the specified object.

USING
Specifies that the audit policy is to be used for the specified object. An existing audit policy must not already be defined for the object (SQLSTATE 5U041). If an audit policy already exists, it must be removed or replaced.

REMOVE
Specifies that the audit policy is to be removed from the specified object. Use of the object will no longer be audited according to the audit policy. The association is deleted from the catalog when the audit policy is removed from the object.

REPLACE
Specifies that the audit policy is to replace an existing audit policy for the specified object. This combines both REMOVE and USING options into one

step to ensure that there is no period of time in which an audit policy does not apply to the specified object. If a policy was not in use for the specified object, REPLACE is equivalent to USING.

POLICY *policy-name*
Specifies the audit policy that is to be used to determine audit settings. The *policy-name* must identify an existing audit policy at the current server (SQLSTATE 42704).

Rules

- An AUDIT-exclusive SQL statement must be followed by a COMMIT or ROLLBACK statement (SQLSTATE 5U021). AUDIT-exclusive SQL statements are:
  - AUDIT
  - CREATE AUDIT POLICY, ALTER AUDIT POLICY, or DROP (AUDIT POLICY)
  - DROP (ROLE or TRUSTED CONTEXT if it is associated with an audit policy)
- An AUDIT-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.
- An object can be associated with no more than one policy (SQLSTATE 5U042).

Notes

- Changes are written to the catalog, but do not take effect until after a COMMIT statement executes.
- Changes do not take effect until the next unit of work that references the object to which the audit policy applies. For example, if the audit policy is in use for the database, no current units of work will begin auditing according to the policy until after a COMMIT or a ROLLBACK statement completes.
- Views accessing a table that is associated with an audit policy are audited according to the underlying table's policy.
- The audit policy that applies to a table does not apply to a materialized query table (MQT) based on that table. It is recommended that if you associate an audit policy with a table, you also associate that policy with any MQT based on that table. The compiler might automatically use an MQT, even though an SQL statement references the base table; however, the audit policy in use for the base table will still be in effect.
- When a switch user operation is performed within a trusted context, all audit policies are re-evaluated according to the new user, and no policies from the old user are used for the current session. This applies specifically to audit policies associated directly with the user, the user's group or role memberships, and the user's authorities. For example, if the current session was audited because the previous user was a member of an audited role, and the switched-to user is not a member of that role, that policy no longer applies to the session.
- When a SET SESSION USER statement is executed, the audit policies associated with the original user (and that user's group and role memberships and authorities) are combined with the policies that are associated with the user specified in the SET SESSION USER statement. The audit policies associated with the original user are still in effect, as are the policies for the user specified in the SET SESSION USER statement. If multiple SET SESSION USER statements are issued within a session, only the audit policies associated with the original user and the current user are considered.
- If the object with which an audit policy is associated is dropped, the association to the audit policy is removed from the catalog and no longer exists. If that

object is recreated at some later time, the object will not be audited according to the policy that was associated with it when the object was dropped.

Examples

- *Example 1:* Use the audit policy DBAUDPRF to determine the audit settings for the database at the current server.

```
AUDIT DATABASE USING POLICY DBAUDPRF
```

- *Example 2:* Remove the audit policy from the EMPLOYEE table.

```
AUDIT TABLE EMPLOYEE REMOVE POLICY
```

- *Example 3:* Use the audit policy POWERUSERS to determine the audit settings for the authorities SYSADM, DBADM, and SECADM, as well as the group DBAS.

```
AUDIT SYSADM, DBADM, SECADM, GROUP DBAS USING POLICY POWERUSERS
```

- *Example 4:* Replace the audit policy for the role TELLER with the new policy TELLERPRF.

```
AUDIT ROLE TELLER REPLACE POLICY TELLERPRF
```

BEGIN DECLARE SECTION

The BEGIN DECLARE SECTION statement marks the beginning of a host variable declare section.

Invocation

This statement can only be embedded in an application program. It is not an executable statement. It must not be specified in REXX.

Authorization

None required.

Syntax

Description

The BEGIN DECLARE SECTION statement may be coded in the application program wherever variable declarations can appear in accordance with the rules of the host language. It is used to indicate the beginning of a host variable declaration section. A host variable section ends with an END DECLARE SECTION statement.

Rules

- The BEGIN DECLARE SECTION and the END DECLARE SECTION statements must be paired and may not be nested.
- SQL statements cannot be included within the declare section.
- Variables referenced in SQL statements must be declared in a declare section in all host languages other than REXX. Furthermore, the section must appear before the first reference to the variable. Generally, host variables are not declared in REXX with the exception of LOB locators and file reference variables. In this case, they are not declared within a BEGIN DECLARE SECTION.
- Variables declared outside a declare section should not have the same name as variables declared within a declare section.
- LOB data types must have their data type and length preceded with the SQL TYPE IS keywords.

Examples

- *Example 1:* Define the host variables hv_smint (smallint), hv_vchar24 (varchar(24)), hv_double (double), hv_blob_50k (blob(51200)), hv_struct (of structured type "struct_type" as blob(10240)) in a C program.

```
EXEC SQL  BEGIN DECLARE SECTION;
  short  hv_smint;
  struct {
  short hv_vchar24_len;
  char  hv_vchar24_value[24];
 }                              hv_vchar24;
  double hv_double;
  SQL TYPE IS BLOB(50K) hv_blob_50k;
  SQL TYPE IS struct_type AS BLOB(10k) hv_struct;
EXEC SQL  END DECLARE SECTION;
```

- *Example 2:* Define the host variables HV-SMINT (smallint), HV-VCHAR24 (varchar(24)), HV-DEC72 (dec(7,2)), and HV-BLOB-50k (blob(51200)) in a COBOL program.

```
WORKING-STORAGE SECTION.
      EXEC SQL BEGIN DECLARE SECTION  END-EXEC.
01  HV-SMINT              PIC S9(4)       COMP-4.
01  HV-VCHAR24.
    49 HV-VCHAR24-LENGTH  PIC S9(4)       COMP-4.
    49 HV-VCHAR24-VALUE   PIC X(24).
01  HV-DEC72              PIC S9(5)V9(2)  COMP-3.
01 HV-BLOB-50K            USAGE SQL TYPE IS BLOB(50K).
      EXEC SQL END DECLARE SECTION  END-EXEC.
```

- *Example 3:* Define the host variables HVSMINT (smallint), HVVCHAR24 (char(24)), HVDOUBLE (double), and HVBLOB50k (blob(51200)) in a Fortran program.

```
EXEC SQL  BEGIN DECLARE SECTION
  INTEGER*2      HVSMINT
  CHARACTER*24   HVVCHAR24
  REAL*8         HVDOUBLE
  SQL TYPE IS BLOB(50K)  HVBLOB50K
EXEC SQL  END DECLARE SECTION
```

  Note: In Fortran, if the expected value is greater than 254 bytes, then a CLOB host variable should be used.

- *Example 4:* Define the host variables HVSMINT (smallint), HVBLOB50K (blob(51200)), and HVCLOBLOC (a CLOB locator) in a REXX program.

```
DECLARE :HVCLOBLOC  LANGUAGE TYPE CLOB LOCATOR
call sqlexec 'FETCH c1 INTO :HVSMINT, :HVBLOB50K'
```

  Note that the variables HVSMINT and HVBLOB50K were implicitly defined by using them in the FETCH statement.

CALL

The CALL statement calls a procedure or a foreign procedure.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. When invoked using the command line processor, there are some additional rules for specifying arguments of the procedure. For more information, refer to "Using command line SQL statements and XQuery statements".

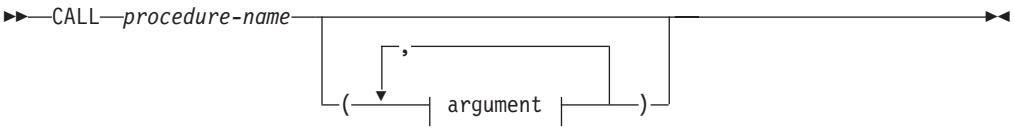
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

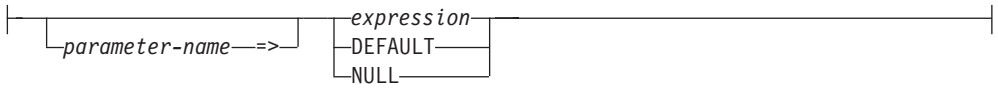
- EXECUTE privilege on the procedure
- DATAACCESS authority

If a matching procedure exists that the authorization ID of the statement is not authorized to execute, an error is returned (SQLSTATE 42501).

Syntax



argument:



Description

procedure-name
Specifies the procedure that is to be called. It must be a procedure that is described in the catalog or that is declared in the scope of the compound SQL (compiled) statement that includes the CALL statement. The specific procedure to invoke is chosen using procedure resolution. (For more details, see the "Notes" section of this statement.)

argument

parameter-name
Name of the parameter to which the argument is assigned. When an argument is assigned to a parameter by name, then all the arguments that follow it must also be assigned by name (SQLSTATE 4274K).

A named argument must be specified only once (implicitly or explicitly) (SQLSTATE 4274K).

Named arguments are not supported on the call to an uncataloged procedure (SQLSTATE 4274K).

expression **or DEFAULT or NULL**
Each specification of *expression*, the DEFAULT keyword, or the NULL keyword is an argument of the CALL. The *n*th unnamed argument of the CALL statement corresponds to the *n*th parameter defined in the CREATE PROCEDURE statement for the procedure.

Named arguments correspond to the same named parameter, regardless of the order in which they are specified.

If the DEFAULT keyword is specified, the default as defined in the CREATE PROCEDURE statement is used if it exists; otherwise the null value is used as the default.

If the NULL keyword is specified, the null value is passed as the parameter value.

Each argument of the CALL must be compatible with the corresponding parameter in the procedure definition as follows:

- IN parameter
  - The argument must be assignable to the parameter.
  - The assignment of a string argument uses the storage assignment rules.
- OUT parameter
  - The argument must be a single variable or parameter marker (SQLSTATE 42886).
  - The argument must be assignable to the parameter.
  - The assignment of a string argument uses the retrieval assignment rules.
- INOUT parameter
  - The argument must be a single variable or parameter marker (SQLSTATE 42886).
  - The argument must be assignable to the parameter.
  - The assignment of a string argument uses the storage assignment rules on invocation and the retrieval assignment rules on return.

Notes

- ***Parameter assignments:*** When the CALL statement is executed, the value of each of its arguments is assigned (using storage assignment) to the corresponding parameter of the procedure. A parameter value that is defined to have a default value can be omitted from the argument list when invoking the procedure.

  When the CALL statement is executed, control is passed to the procedure according to the calling conventions of the host language. When execution of the procedure is complete, the value of each parameter of the procedure is assigned (using storage assignment) to the corresponding argument of the CALL statement defined as OUT or INOUT. If an error is returned by the procedure, OUT arguments are undefined and INOUT arguments are unchanged. For details on the assignment rules, see “Assignments and comparisons”.

  When the CALL statement is in an SQL procedure and is calling another SQL procedure, assignment of XML parameters is done by reference. When an XML argument is passed by reference, the input node trees, if any, are used directly from the XML argument, preserving all properties, including document order, the original node identities, and all parent properties.

- ***Procedure signatures:*** A procedure is identified by its schema, a procedure name, and the number of parameters. This is called a procedure signature, which must be unique within the database. There can be more than one procedure with the same name in a schema, provided that the number of parameters is different for each procedure.
- ***SQL path:*** A procedure can be invoked by referring to a qualified name (schema and procedure name), followed by an optional list of arguments enclosed by parentheses. A procedure can also be invoked without the schema name, resulting in a choice of possible procedures in different schemas with the same number of parameters. In this case, the SQL path is used to assist in procedure resolution. The SQL path is a list of schemas that is searched to identify a procedure with the same name and number of parameters. For static CALL statements, SQL path is specified using the FUNCPATH bind option. For dynamic CALL statements, SQL path is the value of the CURRENT PATH special register.
- ***Procedure resolution:*** Given a procedure invocation, the database manager must decide which of the possible procedures with the same name to execute.

  Local scope procedure resolution is used when a procedure is invoked from within a compound SQL (compiled) statement and either of the following criteria exist:

  – A procedure with the same name as the invoked procedure is declared in the same compound SQL (compiled) statement
  – A procedure with the same name as the invoked procedure is declared in a compound SQL (compiled) statement within which the compound SQL (compiled) statement that invoked the procedure is nested

  Local scope procedure resolution means that only declared procedures within the scope of the compound SQL (compiled) statement that invoked the procedure are considered during procedure resolution regardless of the existence of possible matching built-in procedures, schema procedures, or module procedures. *Global scope procedure resolution* is used in all other cases and considers candidates from schemas and modules depending on the context of the invocation and the qualification of the procedure name.

  – Let A be the number of arguments in a procedure invocation.
  – Let P be the number of parameters in a procedure signature.
  – Let N be the number of parameters without a default.

  Candidate procedures for resolution of a procedure invocation are selected based on the following criteria:

  – Each candidate procedure has a matching name and an applicable number of parameters. An applicable number of parameters satisfies the condition $N \leq A \leq P$.
  – Each candidate procedure has parameters such that for each named argument in the CALL statement there exists a parameter with a matching name that does not already correspond to a positional (or unnamed) argument.
  – Each parameter of a candidate procedure that does not have a corresponding argument in the CALL statement, specified by either position or name, is defined with a default.
  – Each candidate procedure from a set of one or more schemas has the EXECUTE privilege associated with the authorization ID of the statement invoking the function.

  In addition, the set of candidate procedures depends on the environment where the procedure is invoked and how the procedure name is qualified.

- If the procedure name is unqualified, procedure resolution is done using the steps that follow:
  1. If the procedure is invoked from within a compound SQL (compiled) statement and a declared procedure with the same name exists in the nested scope, search the set of compound SQL (compiled) statements within which the CALL statement is nested for candidate procedures. If no candidate procedures are found, an error is returned (SQLSTATE 42884). If a single candidate procedure is found, resolution is complete. If there are multiple candidate procedures, determine the candidate procedure with the lowest number of parameters and eliminate candidate procedures with a higher number of parameters.
  2. If the procedure is invoked from within a module object, search within the module for candidate procedures. If one or more candidate procedures are found in the context module, then these candidate procedures are included with any candidate procedures from the schemas in the SQL path (see next item).
  3. Search all schema procedures with a schema in the SQL path for candidate procedures. If one or more candidate procedures are found in the schemas of the SQL path, then these candidate procedures are included with any candidate procedures from the context module (see previous item). If a single candidate procedure remains, resolution is complete. If there are multiple candidate procedures, choose the procedure from the context module if still a candidate and otherwise choose the procedure whose schema is earliest in the SQL path. If there are still multiple candidate procedures, determine the candidate procedure with the lowest number of parameters and eliminate candidate procedures with a higher number of parameters.

  If there are no candidate procedures remaining after step 3, an error is returned (SQLSTATE 42884).
- If the procedure name is qualified, procedure resolution is done using the steps that follow:
  1. If the procedure is invoked from within a compound SQL (compiled) statement and a declared procedure with the same name exists where the qualifier matches the label of the compound SQL (compiled) statement from the set of compound SQL (compiled) statements within which the CALL statement is nested, search that compound SQL (compiled) statement with the matching label for candidate procedures. If no candidate procedures are found, an error is returned (SQLSTATE 42884). If a single candidate procedure is found, resolution is complete. If there are multiple candidate procedures, determine the candidate procedure with the lowest number of parameters and eliminate candidate procedures with a higher number of parameters.
  2. If the procedure is invoked from within a module and the qualifier matches the name of the module from within which the procedure is invoked, search within the module for candidate procedures. If the qualifier is a single identifier, then the schema name of the module is ignored when matching the module name. If the qualifier is a two part identifier, then it is compared to the schema-qualified module name when determining a match. If a single candidate procedure exists, resolution is complete. If there are multiple candidate procedures, choose the candidate procedure with the least number of parameters. If the qualifier does not match or there are no candidate procedures, then continue with the next step.

3. Consider the qualifier as a schema name and search within that schema for candidate procedures. If a single candidate procedure exists, resolution is complete. If there are multiple candidate procedures, choose the candidate procedure with the least number of parameters and resolution is complete. If the schema does not exist or there are no authorized candidate procedures, and the qualifier matched the name of the module in the first step, then return an error. Otherwise, continue to the next step.
4. Consider the qualifier as a module name, without considering EXECUTE privilege on modules.
   - If the module name is qualified with a schema name, then search published procedures within this module for candidate procedures.
   - If the module name is not qualified with a schema name, then the schema for the module is the first schema in the SQL path that has a matching module name. If found, then search published procedures within this module for candidate procedures.
   - If the module is not found using the SQL path, check for a module public alias that matches the name of the procedure qualifier. If found, then search published procedures within this module for candidate procedures.

   If a matching module is not found or there are no candidate procedures in the matching module, then a procedure not found error is returned (SQLSTATE 42884). If there are multiple candidate procedures, choose the candidate procedure with the least number of parameters. Resolution is complete if the authorization ID of the CALL statement has EXECUTE privilege on the module of the remaining candidate procedure, otherwise an authorization error is returned (SQLSTATE 42501).

- ***Retrieving the DB2_RETURN_STATUS from an SQL procedure:*** If an SQL procedure successfully issues a RETURN statement with a status value, this value is returned in the first SQLERRD field of the SQLCA. If the CALL statement is issued in an SQL procedure, use the GET DIAGNOSTICS statement to retrieve the DB2_RETURN_STATUS value. The value is -1 if the SQLSTATE indicates an error. The values is 0 if no error is returned and the RETURN statement was not specified in the procedure.
- ***Returning result sets from procedures:*** If the calling program is written using CLI, JDBC, or SQLJ, or the caller is an SQL procedure, result sets can be returned directly to the caller. The procedure indicates that a result set is to be returned by declaring a cursor on that result set, opening a cursor on the result set, and leaving the cursor open when exiting the procedure.

  At the end of a procedure:

  – For every cursor that has been left open, a result set is returned to the caller or (for WITH RETURN TO CLIENT cursors) directly to the client.
  – Only unread rows are passed back. For example, if the result set of a cursor has 500 rows, and 150 of those rows have been read by the procedure at the time the procedure is terminated, rows 151 through 500 will be returned to the caller or application (as appropriate).

  If the procedure was invoked from CLI or JDBC, and more than one cursor is left open, the result sets can only be processed in the order in which the cursors were opened.
- ***Improving performance:*** The values of all arguments are passed from the application to the procedure. To improve the performance of this operation, host variables that correspond to OUT parameters and have lengths of more than a few bytes should be set to the null value before the CALL statement is executed.

- *Nesting CALL statements:* Procedures can be called from routines as well as application programs. When a procedure is called from a routine, the call is considered to be nested.

  If a procedure returns any query result sets, the result sets are returned as follows:

  – RETURN TO CALLER result sets are visible only to the program that is at the previous nesting level.
  – RETURN TO CLIENT results sets are visible only if the procedure was invoked from a set of nested procedures. If a function or method occurs anywhere in the call chain, the result set is not visible. If the result set is visible, it is only visible to the client application that made the initial procedure call.

  Consider the following example:

```
Client program:
EXEC SQL CALL PROCA;

  PROCA:
  EXEC SQL CALL PROCB;

    PROCB:
    EXEC SQL DECLARE B1 CURSOR WITH RETURN TO CLIENT ...;
    EXEC SQL DECLARE B2 CURSOR WITH RETURN TO CALLER ...;
    EXEC SQL DECLARE B3 CURSOR FOR SELECT UDFA FROM T1;

      UDFA:
      EXEC SQL CALL PROCC;

        PROCC:
        EXEC SQL DECLARE C1 CURSOR WITH RETURN TO CLIENT ...;
        EXEC SQL DECLARE C2 CURSOR WITH RETURN TO CALLER ...;
```

  From procedure PROCB:

  – Cursor B1 is visible in the client application, but not visible in procedure PROCA.
  – Cursor B2 is visible in PROCA, but not visible to the client.

  From procedure PROCC:

  – Cursor C1 is visible to neither UDFA nor to the client application. (Because UDFA appears in the call chain between the client and PROCC, the result set is not returned to the client.)
  – Cursor C2 is visible in UDFA, but not visible to any of the higher procedures.

- *Nesting procedures within triggers, compound statements, functions, or methods:* When a procedure is called within a trigger, compound statement, function, or method:

  – The procedure must not issue a COMMIT or a ROLLBACK statement.
  – Result sets returned from the procedure cannot be accessed.
  – If the procedure is defined as READS SQL DATA or MODIFIES SQL DATA, no statement in the procedure can access a table that is being modified by the statement that invoked the procedure (SQLSTATE 57053). If the procedure is defined as MODIFIES SQL DATA, no statement in the procedure can modify a table that is being read or modified by the statement that invoked the procedure (SQLSTATE 57053).

    When a procedure is called within a function or method:

    - The procedure has the same table access restrictions as the invoking function or method.

  - Savepoints defined before the function or method was invoked will not be visible to the procedure, and savepoints defined inside the procedure will not be visible outside the function or method.
  - RETURN TO CLIENT result sets returned from the procedure cannot be accessed from the client.

- ***Compilation of CALL statements from DB2 for i and DB2 for z/OS:*** The compilation of CALL statements from DB2 for i and DB2 for z/OS implicitly behave as if CALL_RESOLUTION DEFERRED was specified. When CALL statements are compiled with CALL_RESOLUTION DEFERRED, all arguments must be provided via host variables, and expressions are not allowed.
- ***Syntax alternatives:*** There is an older form of the CALL statement that can be embedded in an application by precompiling the application with the CALL_RESOLUTION DEFERRED option. This option is not available for SQL procedures and federated procedures.

Examples

- *Example 1:* A Java™ procedure is defined in the database using the following statement:

```
CREATE PROCEDURE PARTS_ON_HAND (IN PARTNUM INTEGER,
                                OUT COST DECIMAL(7,2),
                                OUT QUANTITY INTEGER)
               EXTERNAL NAME 'parts!onhand'
               LANGUAGE JAVA
               PARAMETER STYLE DB2GENERAL;
```

  A Java application calls this procedure using the following code fragment:

```
...
CallableStatement stpCall;

String sql = "CALL PARTS_ON_HAND (?, ?, ?)";

stpCall = con.prepareCall(sql); /*con is the connection */

stpCall.setInt(1, hvPartnum);
stpCall.setBigDecimal(2, hvCost);
stpCall.setInt(3, hvQuantity);

stpCall.registerOutParameter(2, Types.DECIMAL, 2);
stpCall.registerOutParameter(3, Types.INTEGER);

stpCall.execute();

hvCost = stpCall.getBigDecimal(2);
hvQuantity = stpCall.getInt(3);
...
```

  This application code fragment will invoke the Java method `onhand` in class `parts`, because the procedure name specified on the CALL statement is found in the database and has the external name `parts!onhand`.

- *Example 2:* There are six FOO procedures, in four different schemas, registered as follows (note that not all required keywords appear):

```
CREATE PROCEDURE AUGUSTUS.FOO (INT) SPECIFIC FOO_1 ...
CREATE PROCEDURE AUGUSTUS.FOO (DOUBLE, DECIMAL(15, 3)) SPECIFIC FOO_2 ...
CREATE PROCEDURE JULIUS.FOO (INT) SPECIFIC FOO_3 ...
CREATE PROCEDURE JULIUS.FOO (INT, INT, INT) SPECIFIC FOO_4 ...
CREATE PROCEDURE CAESAR.FOO (INT, INT) SPECIFIC FOO_5 ...
CREATE PROCEDURE NERO.FOO (INT,INT) SPECIFIC FOO_6 ...
```

  The procedure reference is as follows (where I1 and I2 are INTEGER values):

```
CALL FOO(I1, I2)
```

Assume that the application making this reference has an SQL path established as:

```
"JULIUS", "AUGUSTUS", "CAESAR"
```

Following through the algorithm...

The procedure with specific name FOO_6 is eliminated as a candidate, because the schema "NERO" is not included in the SQL path. FOO_1, FOO_3, and FOO_4 are eliminated as candidates, because they have the wrong number of parameters. The remaining candidates are considered in order, as determined by the SQL path. Note that the types of the arguments and parameters are ignored. The parameters of FOO_5 exactly match the arguments in the CALL, but FOO_2 is chosen because "AUGUSTUS" appears before "CAESAR" in the SQL path.

- *Example 3:* Assume the following procedure exists.

```
CREATE PROCEDURE update_order(
                    IN IN_POID BIGINT,
                    IN IN_CUSTID BIGINT DEFAULT GLOBAL_CUST_ID,
                    IN NEW_STATUS VARCHAR(10) DEFAULT NULL,
                    IN NEW_ORDERDATE DATE DEFAULT NULL,
                    IN NEW_COMMENTS VARCHAR(1000)DEFAULT NULL)...
```

Also assume that the global variable GLOBAL_CUST_ID is set to the value 1002. Call the procedure to change the status of order 5000 for customer 1002 to 'Shipped'. Leave the rest of the order data as it is by allowing the rest of the arguments to default to the null value.

```
CALL update_order (5000, NEW_STATUS => 'Shipped')
```

The customer with ID 1001 has called and indicated that they received their shipment for purchase order 5002 and are satisfied. Update their order.

```
CALL update_order (5002,
    IN_CUSTID => 1001,
    NEW_STATUS => 'Received',
    NEW_COMMENTS => 'Customer satisfied with the order.')
```

- *Example 4:* The following example illustrates procedure resolution, given two procedures named *p1*:

```
CREATE PROCEDURE p1(i1 INT)...
CREATE PROCEDURE p1(i1 INT DEFAULT 0, i2 INT DEFAULT 0)...
CALL p1(i2=>1)
```

Starting with DB2 Version 9.7 Fix Pack 1, the argument names are taken into consideration during the candidate selection process. Therefore, only the second version of *p1* will be considered a candidate. Furthermore, it can be successfully called because *i1* in this version of *p1* is defined with a default, so only specifying *i2* on the call to *p1* is valid.

- *Example 5:* The following example is another illustration of procedure resolution, given two procedures named *p1*:

```
CREATE PROCEDURE p1(i1 INT, i2 INT DEFAULT 0)...
CREATE PROCEDURE p1(i1 INT DEFAULT 0, i2 INT DEFAULT 0, i3 INT DEFAULT 0)...
CALL p1(i2=>1)
```

Starting with DB2 Version 9.7 Fix Pack 1, one of the criteria for a procedure parameter which does not have a corresponding argument in the CALL statement (specified by either position or name) is that the parameter is defined with a default value. Therefore, the first version of *p1* is not considered a candidate.

CASE

The CASE statement selects an execution path based on multiple conditions. This statement should not be confused with the CASE expression, which allows an expression to be selected based on the evaluation of one or more conditions.

Invocation

This statement can be embedded in:

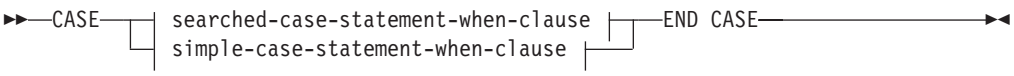
- An SQL procedure definition
- A compound SQL (compiled) statement
- A compound SQL (inlined) statement

The compound SQL statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. The CASE statement is not an executable statement and cannot be dynamically prepared.

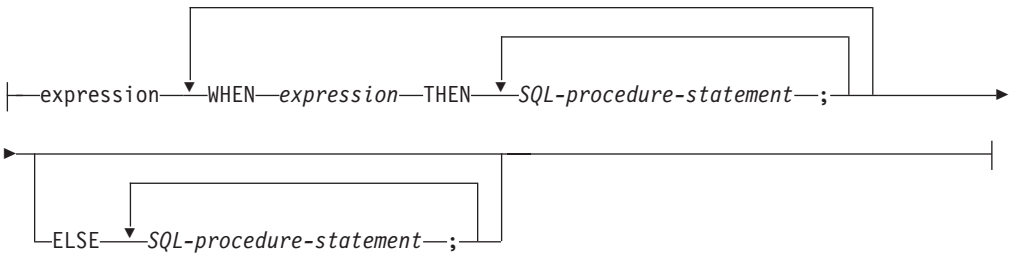
Authorization

No privileges are required to invoke the CASE statement. However, the privileges held by the authorization ID of the statement must include all necessary privileges to invoke the SQL statements and expressions that are embedded in the CASE statement.

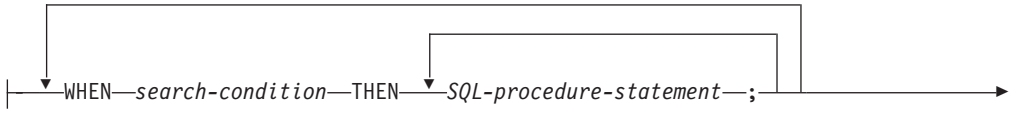
Syntax

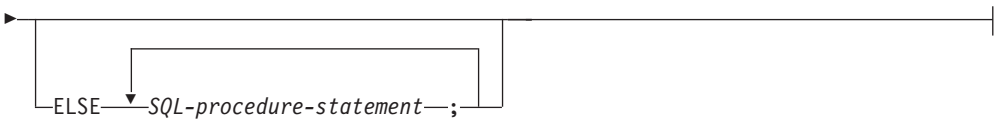


simple-case-statement-when-clause:



searched-case-statement-when-clause:





Description

CASE
Begins a *case-statement*.

simple-case-statement-when-clause
The value of the *expression* before the first WHEN keyword is tested for equality with the value of each *expression* that follows the WHEN keyword. If the search condition is true, the THEN statement is executed. If the result is unknown or false, processing continues to the next search condition. If the result does not match any of the search conditions, and an ELSE clause is present, the statements in the ELSE clause are processed.

searched-case-statement-when-clause
The *search-condition* following the WHEN keyword is evaluated. If it evaluates to true, the statements in the associated THEN clause are processed. If it evaluates to false, or unknown, the next *search-condition* is evaluated. If no *search-condition* evaluates to true and an ELSE clause is present, the statements in the ELSE clause are processed.

SQL-procedure-statement
Specifies a statement that should be invoked. See *SQL-procedure-statement* in "Compound SQL (compiled)" statement.

END CASE
Ends a *case-statement*.

Notes

- If none of the conditions specified in the WHEN are true, and an ELSE clause is not specified, an error is issued at runtime, and the execution of the case statement is terminated (SQLSTATE 20000).
- Ensure that your CASE statement covers all possible execution conditions.

Examples

Depending on the value of SQL variable `v_workdept`, update column DEPTNAME in table DEPARTMENT with the appropriate name.

- *Example 1*: The following example shows how to do this using the syntax for a *simple-case-statement-when-clause*:

```
CASE v_workdept
  WHEN'A00'
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 1';
  WHEN 'B01'
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 2';
  ELSE UPDATE department
    SET deptname = 'DATA ACCESS 3';
END CASE
```

- *Example 2*: The following example shows how to do this using the syntax for a *searched-case-statement-when-clause*:

```
CASE
  WHEN v_workdept = 'A00'
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 1';
  WHEN v_workdept = 'B01'
    THEN UPDATE department
    SET deptname = 'DATA ACCESS 2';
  ELSE UPDATE department
    SET deptname = 'DATA ACCESS 3';
END CASE
```

CLOSE

The CLOSE statement closes a cursor. If a result table was created when the cursor was opened, that table is destroyed.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that cannot be dynamically prepared. When invoked using the command line processor, some options cannot be specified. For more information, refer to "Using command line SQL statements and XQuery statements".

Authorization

If a global variable is referenced, the privileges held by the authorization ID of the statement must include one of the following authorities:

- READ privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

For the authorization required to use a cursor, see "DECLARE CURSOR".

Syntax

Description

cursor-name
Identifies the cursor to be closed. The *cursor-name* must identify a declared cursor as explained in the DECLARE CURSOR statement. When the CLOSE statement is executed, the cursor must be in the open state.

cursor-variable-name
Identifies the cursor to be closed. The *cursor-variable-name* must identify a cursor variable. When the CLOSE statement is executed, the underlying cursor of *cursor-variable-name* must be in the open state (SQLSTATE 24501). A CLOSE statement using *cursor-variable-name* can only be used within a compound SQL (compiled) statement.

WITH RELEASE
The release of all locks that have been held for the cursor is attempted. Note that not all of the locks are necessarily released; these locks may be held for other operations or activities.

Notes

- At the end of a unit of work, all cursors that belong to an application process and that were declared without the WITH HOLD option are implicitly closed.
- An underlying cursor of a cursor variable is implicitly closed when it becomes an orphaned cursor. An underlying cursor becomes orphaned when it is no longer an underlying cursor of any cursor variable. For example, this could occur if all the cursor variables for an underlying cursor are in the same scope and all of them go out of scope at the same time.

- The WITH RELEASE clause has no effect when closing cursors defined in functions or methods. The clause also has no effect when closing cursors defined in procedures called from functions or methods.
- The WITH RELEASE clause has no effect for cursors that are operating under isolation levels CS or UR. When specified for cursors that are operating under isolation levels RS or RR, WITH RELEASE terminates some of the guarantees of those isolation levels. Specifically, if the cursor is opened again, an RS cursor may experience the 'nonrepeatable read' phenomenon and an RR cursor may experience either the 'nonrepeatable read' or 'phantom' phenomenon.

  If a cursor that was originally either RR or RS is reopened after being closed using the WITH RELEASE clause, new locks will be acquired.
- Special rules apply to cursors within a procedure that have not been closed before returning to the calling program.
- While a cursor is open (that is, it has not been closed yet), any changes to sequence values as a result of statements involving that cursor (for example, a FETCH or an UPDATE using the cursor that includes a NEXT VALUE expression for a sequence) will not result in an update to PREVIOUS VALUE for those sequences as seen by that cursor. The PREVIOUS VALUE values for these affected sequences are updated when the cursor is closed explicitly with the CLOSE statement. In a partitioned database environment, if a cursor is closed implicitly by a commit or a rollback, the PREVIOUS VALUE may not be updated with the most recently generated value for the sequence.

Example

A cursor is used to fetch one row at a time into the C program variables `dnum`, `dname`, and `mnum`. Finally, the cursor is closed. If the cursor is reopened, it is again located at the beginning of the rows to be fetched.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM TDEPT
  WHERE ADMRDEPT = 'A00';

EXEC SQL OPEN C1;

while (SQLCODE==0) {                                              .
  EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;
     .
     .
}
EXEC SQL CLOSE C1;
```

COMMENT

The COMMENT statement adds or replaces comments in the catalog descriptions of various objects.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

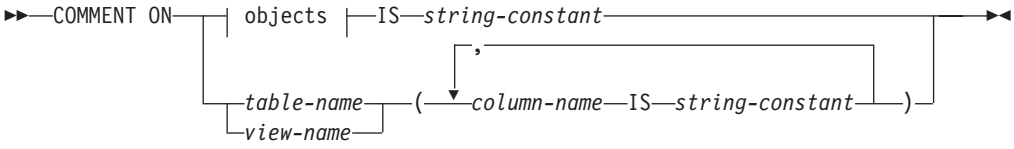
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- Owner of the object (underlying table for column or constraint), as recorded in the OWNER column of the catalog view for the object
- ALTERIN privilege on the schema (applicable only to objects that allow more than one-part names)
- CONTROL privilege on the object (applicable only to index, package, table, or view objects)
- ALTER privilege on the object (applicable only to table objects)
- CREATE_SECURE_OBJECT authority (applicable only to secure functions or secure triggers)
- The WITH ADMIN OPTION (applicable only to roles)
- WLMADM authority (applicable only to workload manager objects)
- SECADM authority (applicable only to audit policy, column mask, role, row permission, secure function, secure trigger, security label, security label component, security policy, or trusted context objects; also applicable to tables for which row level access control or column level access control has been activated)
- DBADM authority (applicable to all objects except audit policy, role, security label, security label component, security policy, or trusted context objects)

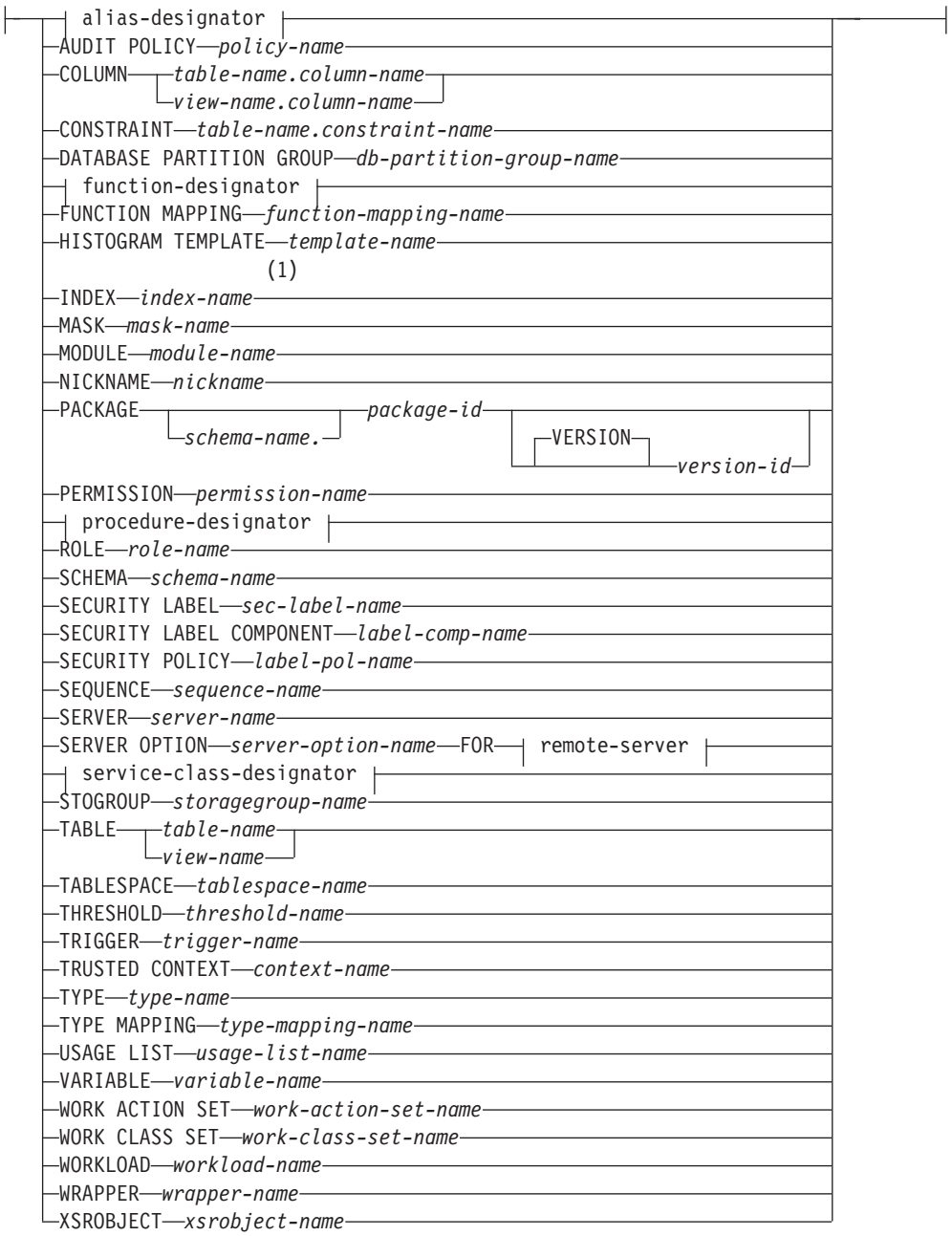
Note that for table space, storage group, or database partition group, and bufferpools, the authorization ID must have SYSCTRL or SYSADM authority.

Syntax

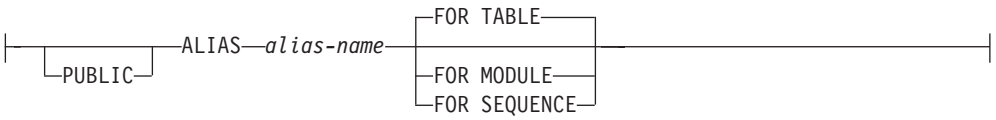


objects:

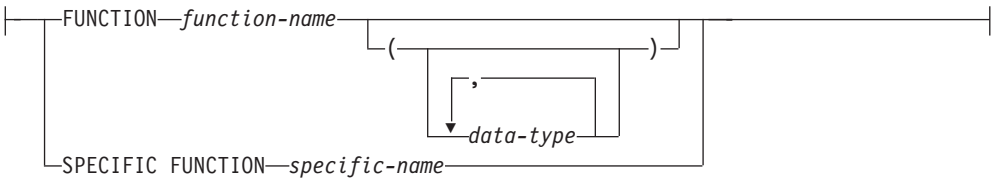
COMMENT



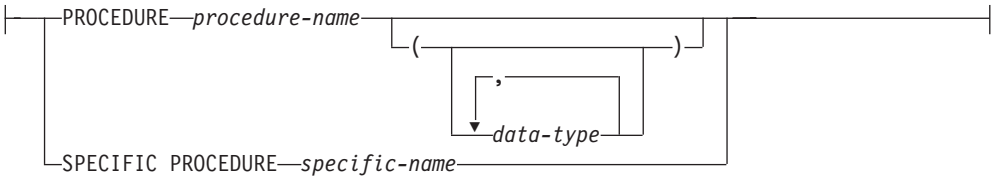
alias-designator:



function-designator:

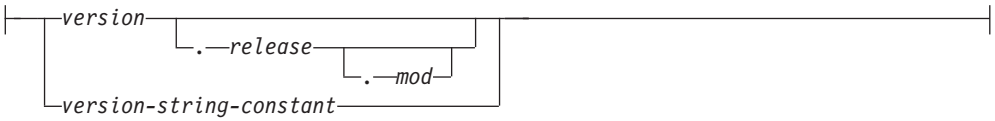


procedure-designator:



remote-server:

server-version:



service-class-designator:

Notes:

1 *Index-name* can be the name of either an index or an index specification.

Description

alias-designator

ALIAS *alias-name*
Indicates a comment will be added or replaced for an alias. The *alias-name* must identify an alias that exists at the current server (SQLSTATE 42704).

FOR TABLE, FOR MODULE, or FOR SEQUENCE
Specifies the object type for the alias.

FOR TABLE
The alias is for a table, view, or nickname. The comment replaces the value of the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the alias.

FOR MODULE
The alias is for a module. The comment replaces the value of the REMARKS column of the SYSCAT.MODULES catalog view for the row that describes the alias.

FOR SEQUENCE
The alias is for a sequence. The comment replaces the value of the REMARKS column of the SYSCAT.SEQUENCES catalog view for the row that describes the alias.

If PUBLIC is specified, the *alias-name* must identify a public alias that exists at the current server (SQLSTATE 42704).

AUDIT POLICY *policy-name*
Indicates a comment will be added or replaced for an audit policy. The *policy-name* must identify an audit policy that exists at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.AUDITPOLICIES catalog view for the row that describes the audit policy.

COLUMN *table-name.column-name* **or** *view-name.column-name*
Indicates that a comment for a column will be added or replaced. The *table-name.column-name* or *view-name.column-name* combination must identify a column and table combination that exists at the current server (SQLSTATE 42704), but must not identify a global temporary table (SQLSTATE 42995). The comment replaces the value of the REMARKS column of the SYSCAT.COLUMNS catalog view for the row that describes the column.

CONSTRAINT *table-name.constraint-name*
Indicates a comment will be added or replaced for a constraint. The *table-name.constraint-name* combination must identify a constraint and the table that it constrains; they must exist at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.TABCONST catalog view for the row that describes the constraint.

DATABASE PARTITION GROUP *db-partition-group-name*
Indicates a comment will be added or replaced for a database partition group. The *db-partition-group-name* must identify a distinct database partition group that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.DBPARTITIONGROUPS catalog view for the row that describes the database partition group.

function-designator
Indicates a comment will be added or replaced for a function. For more information, see "Function, method, and procedure designators" on page 20.

It is not possible to comment on a function that is in the SYSIBM, SYSIBMADM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the function.

FUNCTION MAPPING *function-mapping-name*
Indicates a comment will be added or replaced for a function mapping. The *function-mapping-name* must identify a function mapping that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.FUNCMAPPINGS catalog view for the row that describes the function mapping.

HISTOGRAM TEMPLATE *template-name*
Indicates a comment will be added or replaced for a histogram template. The

template-name must identify a histogram template that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.HISTOGRAMTEMPLATES catalog view for the row that describes the histogram template.

INDEX *index-name*
Indicates a comment will be added or replaced for an index or index specification. The *index-name* must identify either a distinct index or an index specification that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.INDEXES catalog view for the row that describes the index or index specification.

MASK *mask-name*
Identifies the column mask to which the comment applies. *mask-name* must identify a column mask that exists at the current server (SQLSTATE 42704). The comment is placed in the REMARKS column of the SYSCAT.CONTROLS catalog table for the row that describes the mask.

MODULE *module-name*
Indicates a comment will be added or replaced for a module. The *module-name* must identify a module that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.MODULES catalog view for the row that describes the module.

NICKNAME *nickname*
Indicates a comment will be added or replaced for a nickname. The *nickname* must be a nickname that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the nickname.

PACKAGE *schema-name.package-id*
Indicates that a comment will be added or replaced for a package. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The schema name and package ID, together with the implicitly or explicitly specified version ID, must identify a package that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.PACKAGES catalog view for the row that describes the package.

VERSION *version-id*
Identifies which package version is to be commented on. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be commented on in one invocation of the COMMENT statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PERMISSION *permission-name*
Identifies the row permission to which the comment applies. *permission-name* must identify a row permission that exists at the current server (SQLSTATE

42704, SQLCODE -204). The comment is placed in the REMARKS column of the SYSCAT.CONTROLS catalog table for the row that describes the permission.

procedure-designator
Indicates a comment will be added or replaced for a procedure. For more information, see "Function, method, and procedure designators" on page 20.

It is not possible to comment on a procedure that is in the SYSIBM, SYSIBMADM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the procedure.

ROLE *role-name*
Indicates a comment will be added or replaced for a role. The *role-name* must identify a role that exists at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.ROLES catalog view for the row that describes the role.

SCHEMA *schema-name*
Indicates a comment will be added or replaced for a schema. The *schema-name* must identify a schema that exists at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.SCHEMATA catalog view for the row that describes the schema.

SECURITY LABEL *sec-label-name*
Indicates that a comment will be added or replaced for the security label named *sec-label-name*. The name must be qualified with a security policy and must identify a security label that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYLABELS catalog view for the row that describes the security label.

SECURITY LABEL COMPONENT *label-comp-name*
Indicates that a comment will be added or replaced for the security label component named *label-comp-name*. The *label-comp-name* must identify a security label component that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYLABELCOMPONENTS catalog view for the row that describes the security label component.

SECURITY POLICY *label-pol-name*
Indicates that a comment will be added or replaced for the security policy named *label-pol-name*. The *label-pol-name* must identify a security policy that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYPOLICIES catalog view for the row that describes the security policy.

SEQUENCE *sequence-name*
Indicates a comment will be added or replaced for a sequence. The *sequence-name* must identify a sequence that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SEQUENCES catalog view for the row that describes the sequence.

SERVER *server-name*
Indicates a comment will be added or replaced for a data source. The *server-name* must identify a data source that exists at the current server

(SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVERS catalog view for the row that describes the data source.

SERVER OPTION *server-option-name* **FOR** *remote-server*
Indicates a comment will be added or replaced for a server option.

server-option-name
Identifies a server option. This option must be one that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVEROPTIONS catalog view for the row that describes the server option.

remote-server
Describes the data source to which the *server-option* applies.

SERVER *server-name*
Names the data source to which the *server-option* applies. The *server-name* must identify a data source that exists at the current server.

TYPE *server-type*
Specifies the type of data source-for example, DB2 for z/OS or Oracle-to which the *server-option* applies. The *server-type* can be specified in either lower- or uppercase; it will be stored in uppercase in the catalog.

VERSION
Specifies the version of the data source identified by *server-name*.

version
Specifies the version number. *version* must be an integer.

release
Specifies the number of the release of the version denoted by *version*. *release* must be an integer.

mod
Specifies the number of the modification of the release denoted by *release*. *mod* must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release*, and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*
Identifies the wrapper that is used to access the data source referenced by *server-name*.

service-class-designator

SERVICE CLASS *service-class-name*
Indicates a comment will be added or replaced for a service class. The *service-class-name* must identify a service class that exists at the current server (SQLSTATE 42704). To add or replace a comment for a service subclass, the *service-superclass-name* must be specified using the UNDER clause. The comment replaces the value for the REMARKS column of the SYSCAT.SERVICECLASSES catalog view for the row that describes the service class.

UNDER *service-superclass-name*
Specifies the service superclass of the service subclass when adding or

replacing a comment for a service subclass. The *service-superclass-name* must identify a service superclass that exists at the current server (SQLSTATE 42704).

STOGROUP *storagegroup-name*
Indicates a comment will be added or replaced for a storage group. The *storagegroup-name* must identify a distinct storage group that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.STOGROUPS catalog view for the row that describes the storage group.

TABLE *table-name* **or** *view-name*
Indicates a comment will be added or replaced for a table or view. The *table-name* or *view-name* must identify a table or view (not an alias or nickname) that exists at the current server (SQLSTATE 42704) and must not identify a declared temporary table (SQLSTATE 42995). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the table or view.

TABLESPACE *tablespace-name*
Indicates a comment will be added or replaced for a table space. The *tablespace-name* must identify a distinct table space that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLESPACES catalog view for the row that describes the table space.

THRESHOLD *threshold-name*
Indicates a comment will be added or replaced for a threshold. The *threshold-name* must identify a threshold that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.THRESHOLDS catalog view for the row that describes the threshold.

TRIGGER *trigger-name*
Indicates a comment will be added or replaced for a trigger. The *trigger-name* must identify a distinct trigger that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TRIGGERS catalog view for the row that describes the trigger.

TRUSTED CONTEXT *context-name*
Indicates a comment will be added or replaced for a trusted context. The *context-name* must identify a trusted context that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.CONTEXTS catalog view for the row that describes the trusted context.

TYPE *type-name*
Indicates a comment will be added or replaced for a user-defined type. The *type-name* must identify a user-defined type that exists at the current server (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.DATATYPES catalog view for the row that describes the user-defined type.

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

TYPE MAPPING *type-mapping-name*
Indicates a comment will be added or replaced for a user-defined data type

mapping. The *type-mapping-name* must identify a data type mapping that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TYPEMAPPINGS catalog view for the row that describes the mapping.

USAGE LIST *usage-list-name*
Indicates a comment will be added or replaced for a usage list. The *usage-list-name* must identify a usage list that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.USAGELISTS catalog view for the row that describes the usage list.

VARIABLE *variable-name*
Indicates a comment will be added or replaced for a global variable. The *variable-name* must identify a global variable that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.VARIABLES catalog view for the row that describes the variable.

WORK ACTION SET *work-action-set-name*
Indicates a comment will be added or replaced for a work action set. The *work-action-set-name* must identify a work action set that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WORKACTIONSETS catalog view for the row that describes the work action set.

WORK CLASS SET *work-class-set-name*
Indicates a comment will be added or replaced for a work class set. The *work-class-set-name* must identify a work class set that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WORKCLASSSETS catalog view for the row that describes the work class set.

WORKLOAD *workload-name*
Indicates that a comment will be added or replaced for a workload. The *workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WORKLOADS catalog view for the row that describes the workload.

WRAPPER *wrapper-name*
Indicates a comment will be added or replaced for a wrapper. The *wrapper-name* must identify a wrapper that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WRAPPERS catalog view for the row that describes the wrapper.

XSROBJECT *xsrobject-name*
Indicates a comment will be added or replaced for an XSR object. The *xsrobject-name* must identify an XSR object that exists at the current server (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.XSROBJECTS catalog view for the row that describes the XSR object.

IS *string-constant*
Specifies the comment to be added or replaced. The *string-constant* can be any character string constant of up to 254 bytes. (Carriage return and line feed each count as 1 byte.)

table-name|*view-name* **({ *column-name* IS *string-constant* } ...)**
This form of the COMMENT statement provides the ability to specify comments for multiple columns of a table or view. The column names must not be qualified, each name must identify a column of the specified table or view, and the table or view must exist at the current server. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

A comment cannot be made on a column of an inoperative view (SQLSTATE 51024).

Notes

- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
  - DISTINCT TYPE *type-name* can be specified in place of TYPE *type-name*
  - DATA TYPE *type-name* can be specified in place of TYPE *type-name*
  - SYNONYM can be specified in place of ALIAS

Examples

- *Example 1:* Add a comment for the EMPLOYEE table.

```
COMMENT ON TABLE EMPLOYEE
  IS 'Reflects first quarter reorganization'
```

- *Example 2:* Add a comment for the EMP_VIEW1 view.

```
COMMENT ON TABLE EMP_VIEW1
  IS 'View of the EMPLOYEE table without salary information'
```

- *Example 3:* Add a comment for the EDLEVEL column of the EMPLOYEE table.

```
COMMENT ON COLUMN EMPLOYEE.EDLEVEL
  IS 'highest grade level passed in school'
```

- *Example 4:* Add comments for two different columns of the EMPLOYEE table.

```
COMMENT ON EMPLOYEE
  (WORKDEPT IS 'see DEPARTMENT table for names',
  EDLEVEL IS 'highest grade level passed in school' )
```

- *Example 5:* Pellow wants to comment on the CENTRE function, which he created in his PELLOW schema, using the signature to identify the specific function to be commented on.

```
COMMENT ON FUNCTION CENTRE (INT,FLOAT)
  IS 'Frank''s CENTRE fctn, uses Chebychev method'
```

- *Example 6:* McBride wants to comment on another CENTRE function, which she created in the PELLOW schema, using the specific name to identify the function instance to be commented on:

```
COMMENT ON SPECIFIC FUNCTION PELLOW.FOCUS92 IS
  'Louise''s most triumphant CENTRE function, uses the
       Brownian fuzzy-focus technique'
```

- *Example 7:* Comment on the function ATOMIC_WEIGHT in the CHEM schema, where it is known that there is only one function with that name:

```
COMMENT ON FUNCTION CHEM.ATOMIC_WEIGHT
  IS 'takes atomic nbr, gives atomic weight'
```

- *Example 8:* Eigler wants to comment on the SEARCH procedure, which he created in his EIGLER schema, using the signature to identify the specific procedure to be commented on.

```
COMMENT ON PROCEDURE SEARCH (CHAR,INT)
  IS 'Frank''s mass search and replace algorithm'
```

- *Example 9:* Macdonald wants to comment on another SEARCH function, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be commented on:

```
COMMENT ON SPECIFIC PROCEDURE EIGLER.DESTROY IS
  'Patrick''s mass search and destroy algorithm'
```

- *Example 10:* Comment on the procedure OSMOSIS in the BIOLOGY schema, where it is known that there is only one procedure with that name:

```
COMMENT ON PROCEDURE BIOLOGY.OSMOSIS
  IS 'Calculations modelling osmosis'
```

- *Example 11:* Comment on an index specification named INDEXSPEC.

```
COMMENT ON INDEX INDEXSPEC
  IS 'An index specification that indicates to the optimizer
  that the table referenced by nickname NICK1 has an index.'
```

- *Example 12:* Comment on the wrapper whose default name is NET8.

```
COMMENT ON WRAPPER NET8
  IS 'The wrapper for data sources associated with
      Oracle's Net8 client software.'
```

- *Example 13:* Create a comment on the XML schema HR.EMPLOYEE.

```
COMMENT ON XSROBJECT HR.EMPLOYEE
  IS 'This is the base XML Schema for employee data.'
```

- *Example 14:* Create a comment for trusted context APPSERVER.

```
COMMENT ON TRUSTED CONTEXT APPSERVER
  IS 'WebSphere Server'
```

- *Example 15:* Create a comment for column mask M1.

```
COMMENT ON MASK M1 IS 'Column mask for column EMP.SALARY'
```

COMMIT

The COMMIT statement terminates a unit of work and commits the database changes that were made by that unit of work.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax

Description

The unit of work in which the COMMIT statement is executed is terminated and a new unit of work is initiated. All changes made by the following statements executed during the unit of work are committed: ALTER, COMMENT, CREATE, DROP, GRANT, LOCK TABLE, REVOKE, SET INTEGRITY, SET Variable, and the data change statements (INSERT, DELETE, MERGE, UPDATE), including those nested in a query.

The following statements, however, are not under transaction control and changes made by them are independent of the COMMIT statement:

- SET CONNECTION
- SET PASSTHRU

  Note: Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.
- SET SERVER OPTION
- Assignments to updatable special registers

All locks acquired by the unit of work subsequent to its initiation are released, except necessary locks for open cursors that are declared WITH HOLD. All open cursors not defined WITH HOLD are closed. Open cursors defined WITH HOLD remain open, and the cursor is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.) All LOB locators are freed. Note that this is true even when the locators are associated with LOB values retrieved via a cursor that has the WITH HOLD property.

Dynamic SQL statements prepared in a package bound with the `KEEPDYNAMIC YES` option are kept in the SQL context after a COMMIT statement. This is the default behavior. The statement might be implicitly prepared again, as a result of DDL operations that are rolled back within the unit of work. Inactive dynamic SQL

statements prepared in a package bound with KEEPDYNAMIC NO are removed from the SQL context after a COMMIT. The statement must be prepared again before it can be executed in a new transaction.

All savepoints set within the transaction are released.

The following statements behave differently than other data definition language (DDL) and data control language (DCL) statements. Changes made by these statements do not take effect until the statement is committed, even for the current connection that issues the statement. Only one of these statements can be issued by any application at a time, and only one of these statements is allowed within any one unit of work. Each statement must be followed by a COMMIT or a ROLLBACK statement before another one of these statements can be issued.

- CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
- CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
- CREATE WORK ACTION, ALTER WORK ACTION, or DROP (WORK ACTION)
- CREATE WORK CLASS, ALTER WORK CLASS, or DROP (WORK CLASS)
- CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
- GRANT (Workload Privileges) or REVOKE (Workload Privileges)

Notes

- It is strongly recommended that each application process explicitly ends its unit of work before terminating. If the application program ends normally without a COMMIT or ROLLBACK statement then the database manager attempts a commit or rollback depending on the application environment.
- For information about the impact of COMMIT on cached dynamic SQL statements, see “EXECUTE”.
- For information about potential impacts of COMMIT on created temporary tables, see “CREATE GLOBAL TEMPORARY TABLE”.
- For information about potential impacts of COMMIT on declared temporary tables, see “DECLARE GLOBAL TEMPORARY TABLE”.
- The following dynamic SQL statements may be active during COMMIT:
  - Open WITH HOLD cursor
  - COMMIT statement
  - CALL statements under which the COMMIT statement was executed

Example

Commit alterations to the database made since the last commit point.

```
COMMIT WORK
```

Compound SQL

A compound SQL statement is a sequence of individual SQL statements enclosed by BEGIN and END keywords.

There are three types of compound SQL statements:

- Inlined: A compound SQL (inlined) statement is a compound SQL statement that is inlined at run time within another SQL statement. Compound SQL (inlined) statements have the property of being atomically executed; if the execution of any of the statements raises an error, the full statement is rolled back.
- Embedded: Combines one or more other SQL statements (*sub-statements*) into an executable block.
- Compiled: A sequence of SQL statements that executes with a local scope for variables, conditions, cursors, and handlers.

Compound SQL (inlined)

A compound SQL (inlined) statement is a compound SQL statement that is inlined at run time within another SQL statement. Compound SQL (inlined) statements have the property of being atomically executed; if the execution of any of the statements raises an error, the full statement is rolled back.

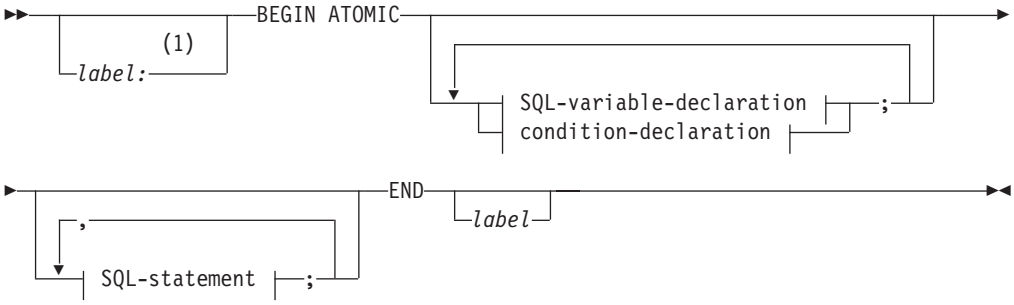
Invocation

This statement can be embedded in a trigger, SQL function, or SQL method, or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

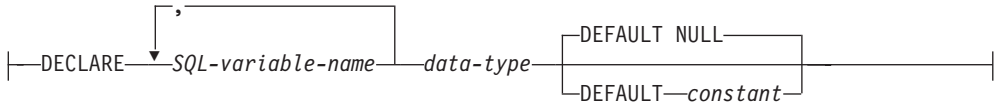
Authorization

The privileges held by the authorization ID of the statement must also include all of the privileges necessary to invoke the SQL statements that are specified in the compound statement.

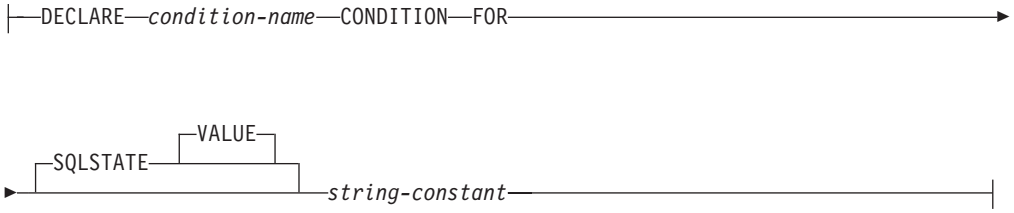
Syntax



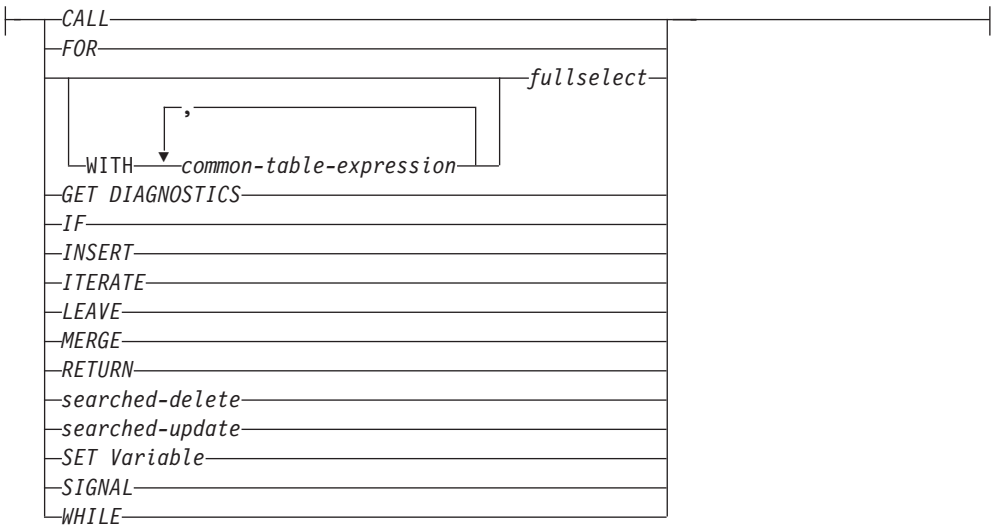
SQL-variable-declaration:



condition-declaration:



SQL-statement:



Notes:

1 A label can only be specified when the statement is in a function, method, or trigger definition.

Description

label
Defines the label for the code block. If the beginning label is specified, it can be used to qualify SQL variables declared in the compound SQL (inlined) statement and can also be specified on a LEAVE statement. If the ending label is specified, it must be the same as the beginning label.

ATOMIC
ATOMIC indicates that, if an error occurs in the compound statement, all SQL statements in the compound statement will be rolled back, and any remaining SQL statements in the compound statement are not processed.

If the ATOMIC keyword is specified in an SQL function in a module or an SQL procedure, the compound statement is processed as a compound SQL (compiled) statement.

SQL-statement
Specifies an SQL statement to be executed within the compound SQL (inlined) statement.

SQL-variable-declaration
Declares a variable that is local to the compound SQL (inlined) statement.

SQL-variable-name
Defines the name of a local variable. DB2 databases convert all SQL variable names to uppercase. The name cannot be the same as:

- Another SQL variable within the compound statement
- A parameter name

If an SQL statement contains an identifier with the same name as an SQL variable and a column reference, the identifier is interpreted as a column.

data-type
Specifies the data type of the variable. The XML data type is not supported in a compound SQL (inlined) statement used in a trigger, in a method, or

as a stand-alone statement (SQLSTATE 429BB). The XML data type is supported when the compound SQL (inlined) statement is used in an SQL function body.

DEFAULT
Defines the default for the SQL variable. The variable is initialized when the compound SQL (inlined) statement is executed. The default value must be assignment-compatible with the data type of the variable. If a default value is not specified, the default for the SQL variable is initialized to the null value.

NULL
Specifies NULL as the default for the SQL variable.

constant
Specifies a constant as the default for the SQL variable.

condition-declaration
Declares a condition name and corresponding SQLSTATE value.

condition-name
Specifies the name of the condition. The condition name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A condition name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement (SQLSTATE 42737).

FOR SQLSTATE *string-constant*
Specifies the SQLSTATE associated with the condition. The *string-constant* must be specified as five characters enclosed by single quotation marks, and the SQLSTATE class (the first two characters) must not be '00'.

Notes

- Compound SQL (inlined) statements are compiled as one single statement. This statement is effective for short scripts involving little control flow logic but significant data flow. For larger constructs with requirements for nested control flow or condition handling, a better choice is to use the compound SQL (compiled) statement or an SQL procedure.
- A procedure called within a compound statement must not issue a COMMIT or a ROLLBACK statement (SQLSTATE 42985).
- ***Table access restrictions:*** If a procedure is defined as READS SQL DATA or MODIFIES SQL DATA, no statement in the procedure can access a table that is being modified by the compound statement that invoked the procedure (SQLSTATE 57053). If the procedure is defined as MODIFIES SQL DATA, no statement in the procedure can modify a table that is being read or modified by the compound statement that invoked the procedure (SQLSTATE 57053).
- ***XML assignments:*** Assignment to parameters and variables of data type XML is done by reference in SQL function bodies.

  When XML values are passed by reference, any input node trees are used directly. This direct usage preserves all properties, including document order, the original node identities, and all parent properties.
- ***Isolation level:*** If a *select-statement*, *fullselect*, or *subselect* specifies an *isolation-clause*, the clause is ignored and a warning is returned.

Example

This example illustrates how inline SQL PL can be used in a data warehousing scenario for data cleansing.

The example introduces three tables. The TARGET table contains the cleansed data. The EXCEPT table stores rows that cannot be cleansed (exceptions) and the SOURCE table contains the raw data to be cleansed.

A simple SQL function called DISCRETIZE is used to classify and modify the data. It returns the null value for all bad data. The compound SQL (inlined) statement then cleanses the data. It walks all rows of the SOURCE table in a FOR-loop and decides whether the current row gets inserted into the TARGET or the EXCEPT table, depending on the result of the DISCRETIZE function. More elaborate mechanisms (multistage cleansing) are possible with this technique.

The same code can be written using an SQL Procedure or any other procedure or application in a host language. However, the compound SQL (inlined) statement offers a unique advantage in that the FOR-loop does not open a cursor and the single row inserts are not really single row inserts. In fact, the logic is effectively a multi-table insert from a shared select.

This is achieved by compilation of the compound SQL (inlined) statement as a single statement. Similar to a view whose body is integrated into the query that uses it and then is compiled and optimized as a whole within the query context, the DB2 optimizer compiles and optimizes both the control and data flow together. The whole logic is therefore executed within the runtime environment of the DB2 database. No data is moved outside of the core DB2 engine, as would be done for a procedure.

The first step is to create the required tables:

```
CREATE TABLE TARGET
  (PK INTEGER NOT NULL
  PRIMARY KEY, C1 INTEGER)
```

This creates a table called TARGET to contain the cleansed data.

```
CREATE TABLE EXCEPT
  (PK INTEGER NOT NULL
  PRIMARY KEY, C1 INTEGER)
```

This creates a table called EXCEPT to contain the exceptions.

```
CREATE TABLE SOURCE
  (PK INTEGER NOT NULL
  PRIMARY KEY, C1 INTEGER)
```

This creates a table called SOURCE to hold the data that is to be cleansed.

Next, a function named DISCRETIZE is created to cleanse the data by throwing out all values outside [0..1000] and aligning them to steps of 10.

```
CREATE FUNCTION DISCRETIZE(RAW INTEGER) RETURNS INTEGER
  RETURN CASE
    WHEN RAW < 0 THEN CAST(NULL AS INTEGER)
    WHEN RAW > 1000 THEN NULL
    ELSE ((RAW / 10) * 10) + 5
  END
```

Then the values are inserted:

```
INSERT INTO SOURCE (PK, C1)
  VALUES (1,   -5),
         (2, NULL),
         (3, 1200),
         (4,   23),
         (5,   10),
         (6,  876)
```

Invoke the function:

```
BEGIN ATOMIC
  FOR ROW AS
    SELECT PK, C1, DISCRETIZE(C1) AS D FROM SOURCE
  DO
    IF ROW.D IS NULL THEN
      INSERT INTO EXCEPT VALUES(ROW.PK, ROW.C1);
    ELSE
      INSERT INTO TARGET VALUES(ROW.PK, ROW.D);
    END IF;
  END FOR;
END
```

And test the results:

```
SELECT * FROM EXCEPT ORDER BY 1
PK          C1
----------- -----------
          1          -5
          2           -
          3        1200
  3 record(s) selected.

SELECT * FROM TARGET ORDER BY 1
PK          C1
----------- -----------
          4          25
          5          15
          6         875
  3 record(s) selected.
```

The final step is to clean up:

```
DROP FUNCTION DISCRETIZE
DROP TABLE SOURCE
DROP TABLE TARGET
DROP TABLE EXCEPT
```

Compound SQL (embedded)

Combines one or more other SQL statements (*sub-statements*) into an executable block.

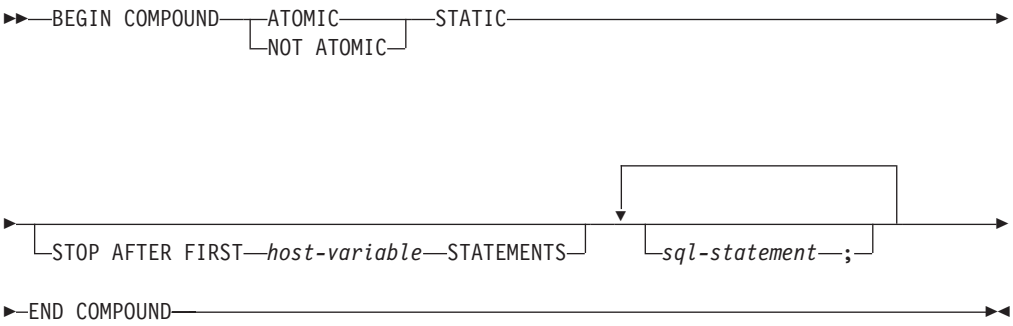
Invocation

This statement can only be embedded in an application program. The entire compound SQL (embedded) statement construct is an executable statement that cannot be dynamically prepared. The statement is not supported in REXX.

Authorization

No privileges are required to invoke an compound SQL (embedded). However, the privileges held by the authorization ID of the statement must include all necessary privileges to invoke the SQL statements that are embedded in the compound statement.

Syntax



Description

ATOMIC
Specifies that, if any of the sub-statements within the compound SQL (embedded) statement fail, then all changes made to the database by any of the sub-statements, including changes made by successful sub-statements, are undone.

NOT ATOMIC
Specifies that, regardless of the failure of any sub-statements, the compound SQL (embedded) statement will not undo any changes made to the database by the other sub-statements.

STATIC
Specifies that input variables for all sub-statements retain their original value. For example, if

```
SELECT ... INTO :abc ...
```

is followed by:

```
UPDATE T1 SET C1 = 5 WHERE C2 = :abc
```

the UPDATE statement will use the value that :abc had at the start of the execution of the compound SQL (embedded) statement, not the value that follows the SELECT INTO.

If the same variable is set by more than one sub-statement, the value of that variable following the compound SQL (embedded) statement is the value set by the last sub-statement.

Note: Non-static behavior is not supported. This means that the sub-statements should be viewed as executing non-sequentially and sub-statements should not have interdependencies.

STOP AFTER FIRST
Specifies that only a certain number of sub-statements will be executed.

host-variable
A small integer that specifies the number of sub-statements to be executed.

STATEMENTS
Completes the STOP AFTER FIRST *host-variable* clause.

sql-statement
All executable statements except the following can be contained within an embedded static compound SQL (embedded) statement:

```
CALL                FETCH
CLOSE               OPEN
CONNECT             PREPARE
Compound SQL        RELEASE (Connection)
DESCRIBE            ROLLBACK
DISCONNECT          SET CONNECTION
EXECUTE IMMEDIATE   SET variable
```

Note: INSERT, UPDATE, and DELETE are not supported in compound SQL for use with nicknames.

If a COMMIT statement is included, it must be the last sub-statement. If COMMIT is in this position, it will be issued even if the STOP AFTER FIRST *host-variable* STATEMENTS clause indicates that not all of the sub-statements are to executed. For example, suppose COMMIT is the last sub-statement in a compound SQL block of 100 sub-statements. If the STOP AFTER FIRST STATEMENTS clause indicates that only 50 sub-statements are to be executed, then COMMIT will be the 51st sub-statement.

An error will be returned if COMMIT is included when using CONNECT TYPE 2 or running in an XA distributed transaction processing environment (SQLSTATE 25000).

Rules

- DB2 Connect™ does not support SELECT statements selecting LOB columns in a compound SQL block.
- No host language code is allowed within a compound SQL (embedded) statement; that is, no host language code is allowed between the sub-statements that make up the compound SQL (embedded) statement.
- Only NOT ATOMIC compound SQL (embedded) statements will be accepted by DB2 Connect.
- Compound SQL (embedded) statements cannot be nested.
- A prepared COMMIT statement is not allowed in an ATOMIC compound SQL (embedded) statement

Notes

- One SQLCA is returned for the entire compound SQL (embedded) statement. Most of the information in that SQLCA reflects the values set by the application server when it processed the last sub-statement. For instance:
  - The SQLCODE and SQLSTATE are normally those for the last sub-statement (the exception is described in the next point).
  - If a 'no data found' warning (SQLSTATE 02000) is returned, that warning is given precedence over any other warning so that a WHENEVER NOT FOUND exception can be acted upon. (This means that the SQLCODE, SQLERRML, SQLERRMC, and SQLERRP fields in the SQLCA that is eventually returned to the application are those from the sub-statement that triggered the 'no data found' warning. If there is more than one 'no data found' warning within the compound SQL (embedded) statement, the fields for the last sub-statement will be the fields that are returned.)
  - The SQLWARN indicators are an accumulation of the indicators set for all sub-statements.
- If one or more errors occurred during NOT ATOMIC compound SQL execution and none of these are of a serious nature, the SQLERRMC will contain information about these errors, up to a maximum of seven errors. The first token of the SQLERRMC will indicate the total number of errors that occurred. The remaining tokens will each contain the ordinal position and the SQLSTATE of the failing sub-statement within the compound SQL (embedded) statement. The format is a character string of the form:

  ```
  nnnXssscccccc
  ```

  with the substring starting with X repeating up to six more times and the string elements defined as follows.

  nnn The total number of statements that produced errors. (If the number would exceed 999, counting restarts at zero.) This field is left-aligned and padded with blanks.

  X The token separator X'FF'.

  sss The ordinal position of the statement that caused the error. (If the number would exceed 999, counting restarts at zero.) For example, if the first statement failed, this field would contain the number one left-aligned ('1 ').

  ccccc The SQLSTATE of the error.
- The second SQLERRD field contains the number of statements that failed (returned negative SQLCODEs).
- The third SQLERRD field in the SQLCA is an accumulation of the number of rows affected by all sub-statements.
- The fourth SQLERRD field in the SQLCA is a count of the number of successful sub-statements. If, for example, the third sub-statement in a compound SQL (embedded) statement failed, the fourth SQLERRD field would be set to 2, indicating that 2 sub-statements were successfully processed before the error was encountered.
- The fifth SQLERRD field in the SQLCA is an accumulation of the number of rows updated or deleted due to the enforcement of referential integrity constraints for all sub-statements that triggered such constraint activity.

Examples

- *Example 1:* In a C program, issue a compound SQL (embedded) statement that updates both the ACCOUNTS and TELLERS tables. If there is an error in any of the statements, undo the effect of all statements (ATOMIC). If there are no errors, commit the current unit of work.

```
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
  UPDATE ACCOUNTS SET ABALANCE = ABALANCE + :delta
    WHERE AID = :aid;
  UPDATE TELLERS SET TBALANCE = TBALANCE + :delta
    WHERE TID = :tid;
  INSERT INTO TELLERS (TID, BID, TBALANCE) VALUES (:i, :branch_id, 0);
  COMMIT;
END COMPOUND;
```

- *Example 2:* In a C program, insert 10 rows of data into the database. Assume the host variable :nbr contains the value 10 and S1 is a prepared INSERT statement. Further, assume that all the inserts should be attempted regardless of errors (NOT ATOMIC).

```
EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC STOP AFTER FIRST :nbr STATEMENTS
  EXECUTE S1 USING DESCRIPTOR :*sqlda0;
  EXECUTE S1 USING DESCRIPTOR :*sqlda1;
  EXECUTE S1 USING DESCRIPTOR :*sqlda2;
  EXECUTE S1 USING DESCRIPTOR :*sqlda3;
  EXECUTE S1 USING DESCRIPTOR :*sqlda4;
  EXECUTE S1 USING DESCRIPTOR :*sqlda5;
  EXECUTE S1 USING DESCRIPTOR :*sqlda6;
  EXECUTE S1 USING DESCRIPTOR :*sqlda7;
  EXECUTE S1 USING DESCRIPTOR :*sqlda8;
  EXECUTE S1 USING DESCRIPTOR :*sqlda9;
END COMPOUND;
```

Compound SQL (compiled)

A sequence of SQL statements that executes with a local scope for variables, conditions, cursors, and handlers.

Invocation

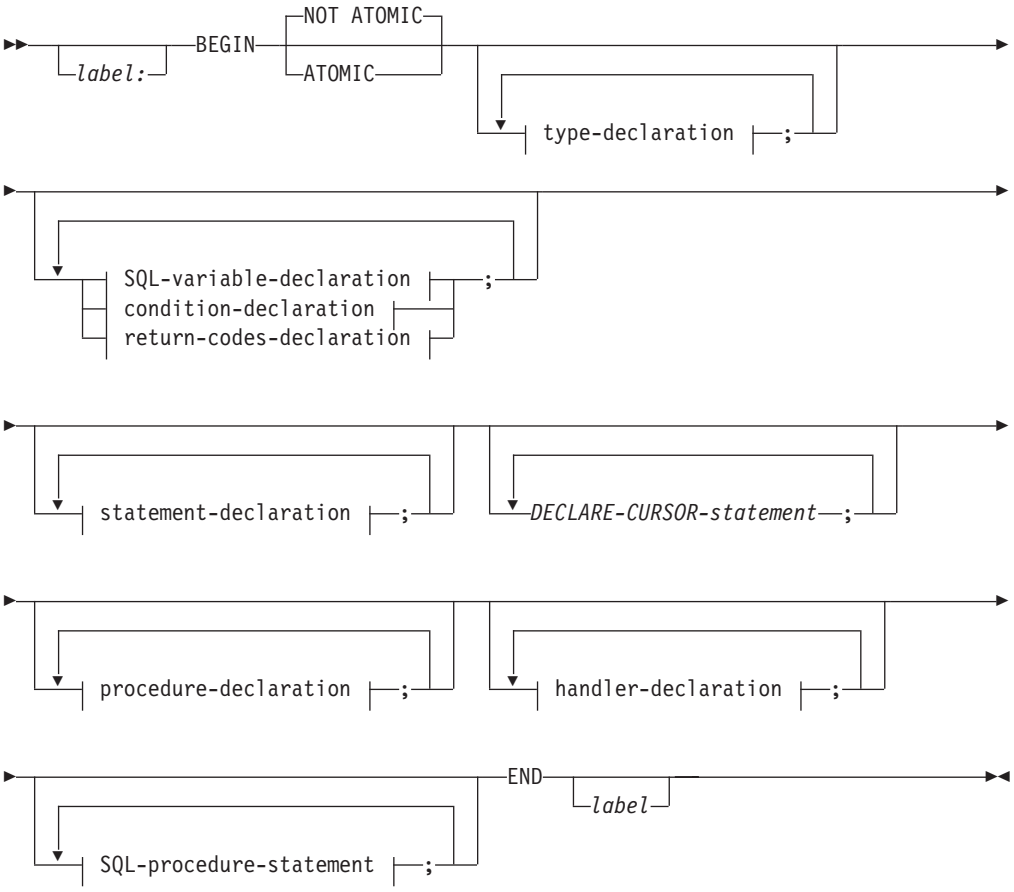
This statement can be embedded in a trigger, SQL function, or SQL procedure; or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

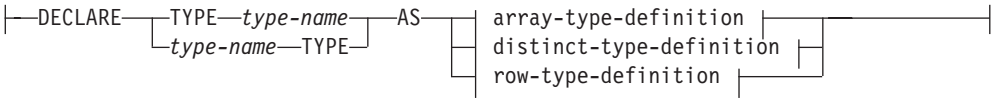
For an *SQL-variable-declaration* that specifies a *cursor-value-constructor* that uses a *select-statement*, the privileges held by the authorization ID of the statement must include the privileges necessary to execute the *select-statement*. See the Authorization section in "SQL queries".

The privileges held by the authorization ID of the statement must also include all of the privileges necessary to invoke the SQL statements that are specified in the compound statement.

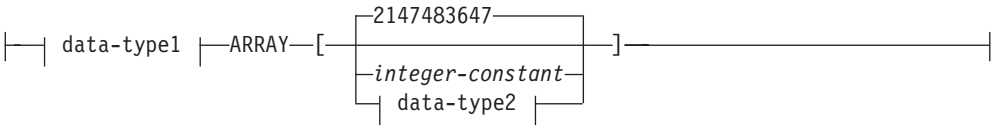
Syntax



type-declaration:

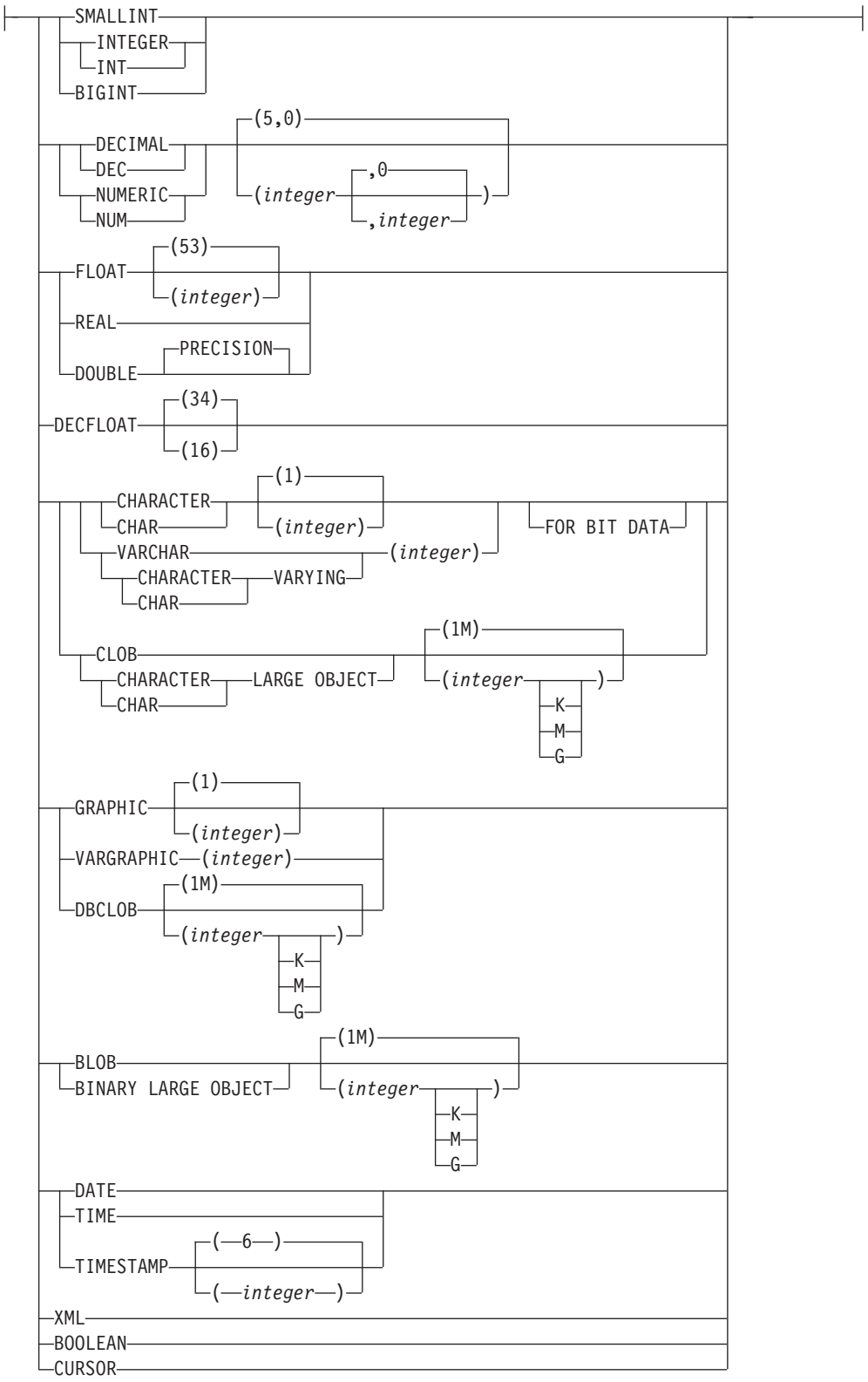


array-type-definition:

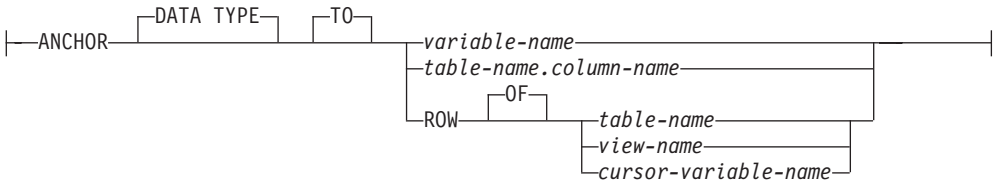


data-type1:

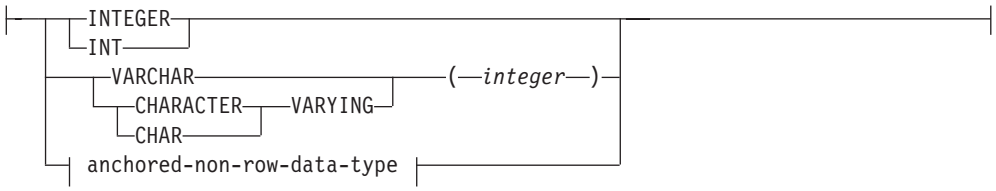
built-in-type:



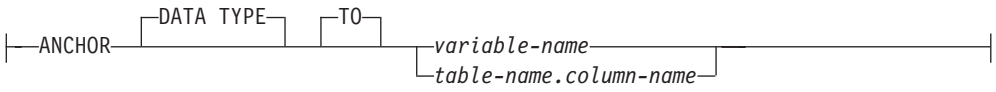
anchored-data-type:



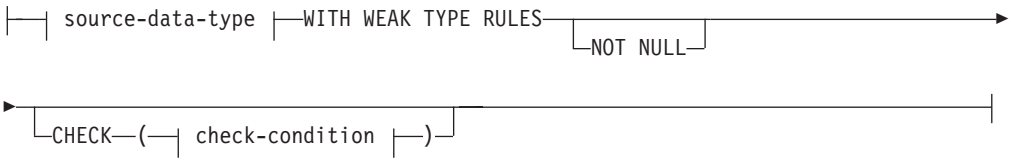
data-type2:



anchored-non-row-data-type:

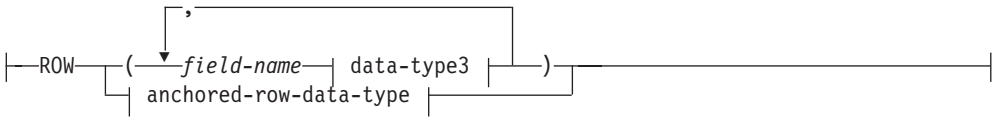


distinct-type-definition:



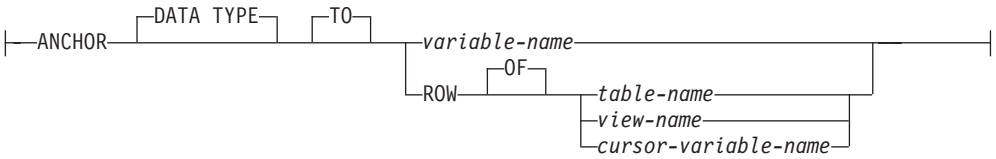
source-data-type:

row-type-definition:

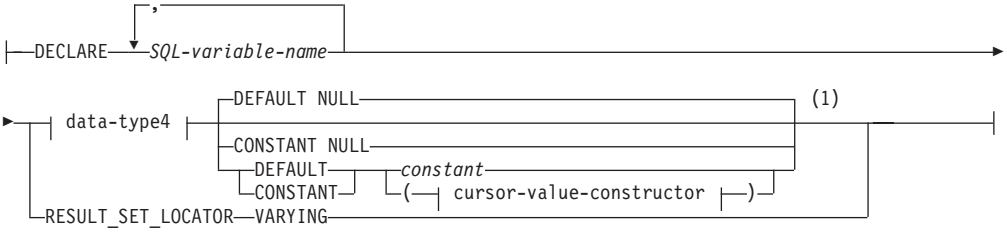


data-type3:

anchored-row-data-type:

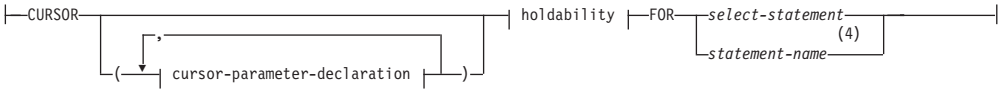


SQL-variable-declaration:

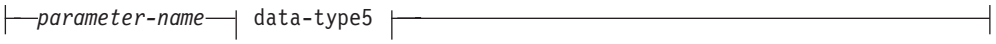


data-type4:

cursor-value-constructor:

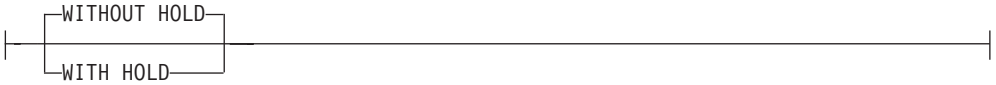


cursor-parameter-declaration:

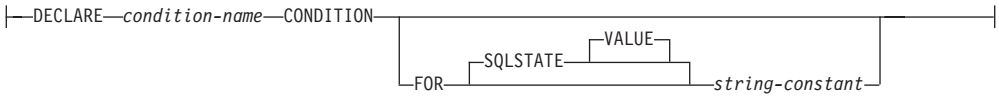


data-type5:

holdability:

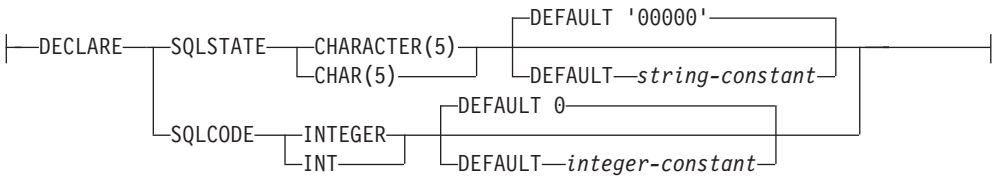


condition-declaration:

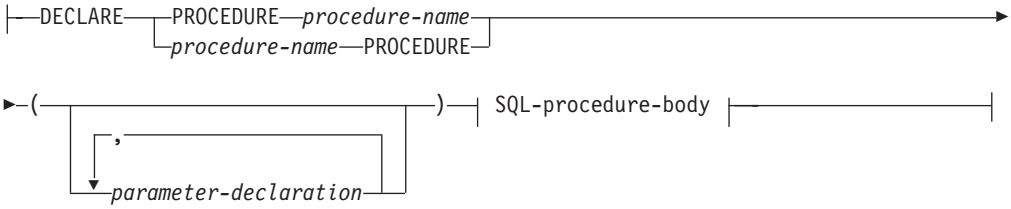


statement-declaration:

return-codes-declaration:

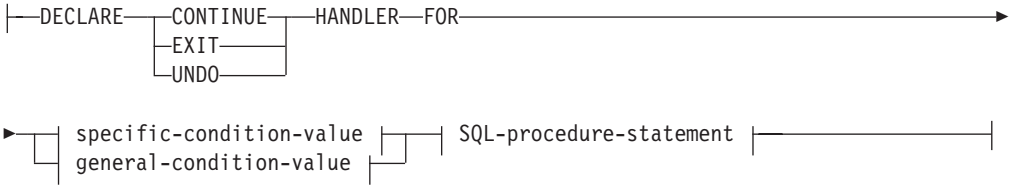


procedure-declaration:

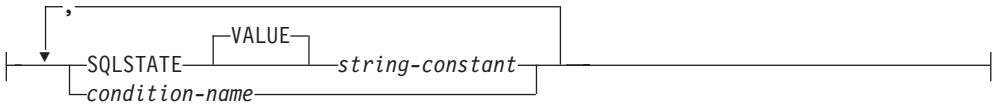


SQL-procedure-body:

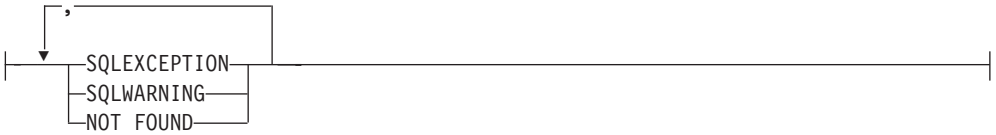
handler-declaration:



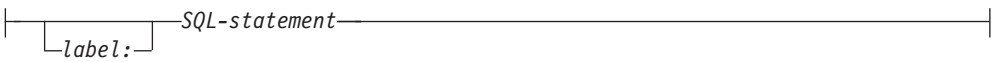
specific-condition-value:



general-condition-value:



SQL-procedure-statement:



Notes:

1 If *data-type4* specifies a **CURSOR** built-in type or *cursor-type-name*, only **NULL** or *cursor-value-constructor* can be specified. Only **DEFAULT NULL** can be explicitly specified for *array-type-name* or *row-type-name* .

2 Only DEFAULT NULL can be explicitly specified for *array-type-name*.

3 Only DEFAULT NULL can be explicitly specified for *row-type-name*.

4 *statement-name* cannot be specified if *cursor-parameter-declaration* is specified.

Description

label
Defines the label for the code block. If the beginning label is specified, it can be used to qualify SQL variables declared in the compound statement and can also be specified on a LEAVE statement. If the ending label is specified, it must be the same as the beginning label.

ATOMIC or NOT ATOMIC
ATOMIC indicates that if an unhandled exception condition occurs in the compound statement, all SQL statements in the compound statement will be rolled back.

NOT ATOMIC indicates that an unhandled exception condition within the compound statement does not cause the compound statement to be rolled back.

If the ATOMIC keyword is specified in a dynamically prepared compound statement or an SQL function that is not within a module, the compound statement is processed as a compound SQL (inlined) statement.

A compound statement that is used in the function body of a module table function can only be defined as NOT ATOMIC.

type-declaration
Declares a user-defined data type that is local to the compound statement.

type-name
Specifies the name of a local user-defined data type. The name cannot be the same as any other type declared within the current compound statement (SQLSTATE 42734). The unqualified *type-name* has the same restrictions as described in any CREATE TYPE statement (SQLSTATE 42939).

array-type-definition
Specifies the attributes of an array data type to associate with the

type-name. See "CREATE TYPE (array)" for a description of the syntax elements. The *row-type-name* can refer to a declared row type that is previously declared and in the scope of the current compound SQL (compiled) statement. The *variable-name* specified in an anchored-data-type clause can refer to a local variable in the scope of the current compound SQL (compiled) statement.

distinct-type-definition
Specifies the source type and optional data type constraints of a weakly typed distinct type to associate with the *type-name*. See "CREATE TYPE (distinct)" for a complete description of the syntax elements. The *variable-name* specified in anchored-non-row-data-type clause can refer to a local variable in the scope of the current compound SQL (compiled) statement. The data type of the anchor *variable-name* or *column-name* must be a built-in data type.

row-type-definition
Specifies the fields of a row data type to associate with the *type-name*. See "CREATE TYPE (row)" for a complete description of the syntax elements. The *variable-name* specified in anchored-non-row-data-type or anchored-row-data-type clauses can refer to a local variable in the scope of the current compound SQL (compiled) statement.

SQL-variable-declaration
Declares a variable that is local to the compound statement.

SQL-variable-name
Defines the name of a local variable. All SQL variable names are converted to uppercase. The name cannot be the same as another SQL variable within the same compound statement and cannot be the same as a parameter name. An SQL variable name must not be the same as a column name. If an SQL statement contains an identifier with the same name as an SQL variable and a column reference, the identifier is interpreted as a column. If the compound statement in which the variable is declared has a label, then references to the variable can be qualified with the label. For example, variable V declared in a compound statement with a label C can be referred to as C.V.

data-type4
Specifies the data type of the variable. A structured type or reference type cannot be specified (SQLSTATE 429BB).

built-in-type
Specifies a built-in data type. For a more complete description of each built-in data type except BOOLEAN and CURSOR, which cannot be specified for a table, see "CREATE TABLE". The XML data type cannot be specified in a compound SQL (compiled) statement used in a trigger, in a function, or as a stand-alone statement (SQLSTATE 429BB). The XML data type can be specified when the compound SQL (compiled) statement is used in an SQL procedure body.

BOOLEAN
For a Boolean.

CURSOR
For a cursor.

anchored-data-type
Identifies another object used to determine the data type of the SQL

variable. The data type of the anchor object has the same limitations that apply to specifying the data type directly, or in the case of a row, to creating a row type.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies an SQL variable, SQL parameter, or global variable. The data type of the referenced variable is used as the data type for *SQL-variable-name*.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for *SQL-variable-name*.

ROW OF *table-name* **or** *view-name*
Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data type of *SQL-variable-name* is an unnamed row type.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following elements (SQLSTATE 428HS):

- An SQL variable or global variable with a strongly typed cursor data type
- An SQL variable or global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a *select-statement* where all the result columns are named.

If the cursor type of the cursor variable is not strongly rtyped using a named row type, the data type of *SQL-variable-name* is an unnamed row type.

array-type-name
Specifies the name of a user-defined array type. The array data type can be a locally declared data type, a schema data type, or a module data type.

cursor-type-name
Specifies the name of a cursor type. The cursor data type can be a schema data type or a module data type.

distinct-type-name
Specifies the name of a distinct type. The distinct data type can be a schema data type or a module data type. The length, precision, and scale of the declared variable are, respectively, the length, precision, and scale of the source type of the distinct type.

row-type-name
Specifies the name of a user-defined row type. The row data type can be a locally declared data type, a schema data type or a module data type. The fields of the variable are the fields of the row type.

DEFAULT or CONSTANT
Specifies a value for the SQL variable when the compound SQL (compiled) statement is referenced. If neither is specified, the default for the SQL variable is the null value. Only DEFAULT NULL can be explicitly specified if *array-type-name* or *row-type-name* is specified.

DEFAULT
Defines the default for the SQL variable. The variable is initialized when the compound SQL (compiled) statement is referenced. The default value must be assignment-compatible with the data type of the variable.

CONSTANT
Specifies that the SQL variable has a fixed value that cannot be changed. An SQL variable that is defined using CONSTANT cannot be used as the target of any assignment operation. The fixed value must be assignment-compatible with the data type of the variable.

NULL
Specifies NULL as the default for the SQL variable.

constant
Specifies a constant as the default for the SQL variable. If *data-type4* specifies a CURSOR built-in type or *cursor-type-name*, *constant* cannot be specified (SQLSTATE 42601).

cursor-value-constructor
A *cursor-value-constructor* specifies the *select-statement* that is associated with the SQL variable. The assignment of a *cursor-value-constructor* to a cursor variable defines the underlying cursor of that cursor variable.

(cursor-parameter-declaration, ...)
Specifies the input parameters of the cursor, including the name and the data type of each parameter. Named input parameters can be specified only if *select-statement* is also specified in *cursor-value-constructor* (SQLSTATE 428HU).

parameter-name
Names the cursor parameter for use as an SQL variable within *select-statement*. The name cannot be the same as any other parameter name for the cursor. Names should also be chosen to avoid any column names that could be used in *select-statement*, since column names are resolved before parameter names.

data-type5
Specifies the data type of the cursor parameter used within *select-statement*. Structured types, and reference types cannot be specified (SQLSTATE 429BB).

built-in-type
Specifies a built-in data type. For a more complete description of each built-in data type, see "CREATE TABLE". The BOOLEAN and CURSOR built-in types cannot be specified (SQLSTATE 429BB).

anchored-non-row-data-type
Identifies another object used to determine the data type of the cursor parameter. The data type of the anchor object has the same limitations that apply to specifying the data type directly.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies a local SQL variable, an SQL parameter, or a global variable. The data type of the referenced variable is used as the data type for the cursor parameter.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for the cursor parameter.

distinct-type-name
Specifies the name of a distinct type. If *distinct-type-name* is specified without a schema name, the distinct type is resolved by searching the schemas in the SQL path.

holdability
Specifies whether the cursor is prevented from being closed as a consequence of a commit operation. See "DECLARE CURSOR" for more information. The default is WITHOUT HOLD.

WITHOUT HOLD
Does not prevent the cursor from being closed as a consequence of a commit operation.

WITH HOLD
Maintains resources across multiple units of work. Prevents the cursor from being closed as a consequence of a commit operation.

select-statement
Specifies the SELECT statement of the cursor. See "select-statement" for more information. If *cursor-parameter-declaration* is included in *cursor-value-constructor*, then *select-statement* must not include any local SQL variables or routine SQL parameters (SQLSTATE 42704).

statement-name
Specifies the prepared *select-statement* of the cursor. See "PREPARE" for an explanation of prepared statements. The target cursor variable must not have a data type that is a strongly rtyped user-defined cursor type (SQLSTATE 428HU). Named input parameters must not be specified in *cursor-value-constructor* if *statement-name* is specified (SQLSTATE 428HU).

RESULT_SET_LOCATOR VARYING
Specifies the data type for a result set locator variable.

condition-declaration
Declares a condition name with an optional associated SQLSTATE value.

condition-name
Specifies the name of the condition. The condition name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A condition name can only be

referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement (SQLSTATE 42737).

CONDITION FOR SQLSTATE VALUE*string-constant*
Specifies the SQLSTATE that is associated with the condition. The string constant must be specified as five characters enclosed in single quotation marks, and the SQLSTATE class (the first two characters) must not be '00'. If this clause is not specified, the condition has no associated SQLSTATE value.

statement-declaration
Declares a list of one or more names that are local to the compound statement. Each name in *statement-name* must not be the same as any other statement name declared in the same compound statement.

return-codes-declaration
Declares special variables called SQLSTATE and SQLCODE that are set automatically to the value returned after processing an SQL statement. Both the SQLSTATE and SQLCODE variables can only be declared in the outermost compound statement when there are nested compound SQL (compiled) statements; for example in an SQL procedure body. These variables may be declared only once per SQL procedure.

declare-cursor-statement
Declares a built-in cursor in the procedure body. Variables of user-defined cursor data types are declared using *SQL-variable-declaration* statements.

Each declared cursor must have a unique name within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). The cursor can be referenced only from within the compound statement in which it is declared, including any compound statements that are nested within that compound statement (SQLSTATE 34000).

Use an OPEN statement to open the cursor, and a FETCH statement to read rows using the cursor. To return result sets from the SQL procedure to the client application, the cursor must be declared using the WITH RETURN clause. The following example returns one result set to the client application:

```
CREATE PROCEDURE RESULT_SET()
  LANGUAGE SQL
  RESULT SETS 1
  BEGIN
    DECLARE C1 CURSOR WITH RETURN FOR
      SELECT id, name, dept, job
        FROM staff;
    OPEN C1;
END
```

Note: To process result sets, you must write your client application using one of the DB2 Call Level Interface (DB2 Call Level Interface), Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), or embedded SQL for Java (SQLJ) application programming interfaces.

For more information about declaring a cursor, see "DECLARE CURSOR".

procedure-declaration
Declares a procedure that is local to the compound statement. The definition of a local procedure does not include the specification of any of the options possible in a “CREATE PROCEDURE (SQL)” statement. The options default as they would for a “CREATE PROCEDURE (SQL)” statement with the exception

of MODIFIES SQL DATA. The data access level for the procedure is automatically determined to be the minimum level required to process the SQL procedure body.

procedure-name
Defines the names of a local procedure. The name must be specified without any qualification (SQLSTATE 42601). The procedure signature, consisting of the *procedure-name* and the number of declared parameters, must be unique within the current compound statement. Outer compound statements within which the current compound statement is nested cannot contain a procedure with the same name.

parameter-declaration
Specifies the parameters of the local procedure. See "CREATE PROCEDURE (SQL)" for a description of the syntax elements. The parameter data type can be a locally declared data type in the scope of the current compound statement.

SQL-procedure-body
Specifies the SQL statement that is the body of the SQL procedure. Names referenced in the *SQL-procedure-body* can refer to declared objects (such as declared variables, data types, and procedures) that are previously declared and in the scope of the compound statement in which the local procedure is declared.

handler-declaration
Specifies a *handler*, and a set of one or more *SQL-procedure-statements* to execute when an exception or completion condition occurs in the compound statement.*SQL-procedure-statement* is a statement that executes when the handler receives control.

A handler is said to be active for the duration of the execution of the set of *SQL-procedure-statements* that follow the set of *handler-declarations* within the compound statement in which the handler is declared, including any nested compound statements.

There are three types of condition handlers:

CONTINUE
After the handler is invoked successfully, control is returned to the SQL statement that follows the statement that raised the exception. If the error that raised the exception is a FOR, IF, CASE, WHILE, or REPEAT statement (but not an SQL-procedure-statement within one of these), then control returns to the statement that follows END FOR, END IF, END CASE, END WHILE, or END REPEAT.

EXIT
After the handler is invoked successfully, control is returned to the end of the compound statement that declared the handler.

UNDO
Before the handler is invoked, any SQL changes that were made in the compound statement are rolled back. After the handler is invoked successfully, control is returned to the end of the compound statement that declared the handler. If UNDO is specified, the compound statement where the handler is declared must be ATOMIC.

The conditions that cause the handler to be activated are defined in the handler-declaration as follows:

specific-condition-value
Specifies that the handler is a *specific condition handler*.

SQLSTATE VALUE*string-constant*
Specifies an SQLSTATE for which the handler is invoked. The first two characters of the SQLSTATE value must not be '00'.

condition-name
Specifies a condition name for which the handler is invoked. The condition name must be previously defined in a condition declaration or it must identify a condition that exists at the current server.

general-condition-value
Specifies that the handler is a *general condition handler*.

SQLEXCEPTION
Specifies that the handler is invoked when an exception condition occurs. An exception condition is represented by an SQLSTATE value whose first two characters are not '00', '01', or '02'.

SQLWARNING
Specifies that the handler is invoked when a warning condition occurs. A warning condition is represented by an SQLSTATE value whose first two characters are '01'.

NOT FOUND
Specifies that the handler is invoked when a NOT FOUND condition occurs. A NOT FOUND condition is represented by an SQLSTATE value whose first two characters are '02'.

SQL-procedure-statement
Specifies the SQL procedure statement.

label
Specifies a label for the SQL procedure statement. The label must be unique within a list of SQL procedure statements, including any compound statements nested within the list. Note that compound statements that are not nested can use the same label. A list of SQL procedure statements is possible in a number of SQL control statements.

SQL-statement
All executable SQL statements except for:

- ALTER
- CONNECT
- CREATE
- DESCRIBE
- DISCONNECT
- DROP
- FLUSH EVENT MONITOR
- GRANT
- REFRESH TABLE
- RELEASE (connection only)
- RENAME TABLE
- RENAME TABLESPACE
- REVOKE
- SET CONNECTION
- SET INTEGRITY

- SET PASSTHRU
- SET SERVER OPTION
- TRANSFER OWNERSHIP

The following executable statements are not supported in stand-alone compound SQL (compiled) statements, but are supported in compound SQL (compiled) statements used within an SQL function, SQL procedure, or trigger:

- CREATE of an index, table, or view
- DROP of an index, table, or view
- GRANT
- ROLLBACK

The ROLLBACK statement is also not supported in any nested statement invoked within the stand-alone compound SQL (compiled) statement.

The following statements, which are not executable statements, are supported in compound SQL (compiled) statements:

- ALLOCATE CURSOR
- ASSOCIATE LOCATORS

Rules

- ATOMIC compound statements cannot be nested.
- The following rules apply to handler declarations:
  - A handler declaration cannot contain the same *condition-name* or SQLSTATE value more than once, and cannot contain an SQLSTATE value and a *condition-name* that represent the same SQLSTATE value.
  - Where two or more condition handlers are declared in a compound statement:
    - No two handler declarations may specify the same general condition category (SQLEXCEPTION, SQLWARNING, NOT FOUND).
    - No two handler declarations may specify the same specific condition, either as an SQLSTATE value or as a *condition-name* that represents the same value.
  - A handler is activated when it is the most appropriate handler for an exception or completion condition. The most appropriate handler is determined based on the following considerations:
    - The scope of a handler declaration *H* is the list of *SQL-procedure-statement* that follows the handler declarations contained within the compound statement in which *H* appears. This means that the scope of *H* does not include the statements contained in the body of the condition handler *H*, implying that a condition handler cannot handle conditions that arise inside its own body. Similarly, for any two handlers *H1* and *H2* declared in the same compound statement, *H1* will not handle conditions arising in the body of *H2*, and *H2* will not handle conditions arising in the body of *H1*.
    - A handler for a *specific-condition-value* or a *general-condition-value* C declared in an inner scope takes precedence over another handler for C declared in an enclosing scope.
    - When a specific handler for condition *C* and a general handler which would also handle *C* are declared in the same scope, the specific handler takes precedence over the general handler.

- When a handler for a module condition that has no associated SQLSTATE value and a handler for SQLSTATE 45000 are declared in the same scope, the handler for the module condition takes precedence over the handler for SQLSTATE 45000.

If an exception condition occurs for which there is no appropriate handler, the SQL procedure containing the failing statement is terminated with an unhandled exception condition. If a completion condition occurs for which there is no appropriate handler, execution continues with the next SQL statement.

- Referencing variables or parameters of data type XML in SQL procedures after a commit or rollback operation occurs, without first assigning new values to these variables, is not supported (SQLSTATE 560CE).
- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.
- If named parameter markers are used in a compound SQL (compiled) statement that is dynamically prepared or executed, every parameter marker name must be unique (SQLSTATE 42997).

Notes

- ***XML assignments:*** Assignment to parameters and variables of data type XML is done by reference.

  Passing parameters of data type XML in a CALL statement to an SQL procedure is done by reference. When XML values are passed by reference, any input node trees are used directly from the XML argument. This direct usage preserves all properties, including document order, the original node identities, and all parent properties.

Examples

- *Example 1*: Create a procedure with a compound SQL (compiled) statement that performs the following actions:
  1. Declares SQL variables
  2. Declares a cursor to return the salary of employees in a department determined by an IN parameter. In the SELECT statement, casts the data type of the *salary* column from a DECIMAL into a DOUBLE.
  3. Declares an EXIT handler for the condition NOT FOUND (end of file) which assigns the value '6666' to the OUT parameter `medianSalary`
  4. Select the number of employees in the given department into the SQL variable `numRecords`
  5. Fetch rows from the cursor in a WHILE loop until 50% + 1 of the employees have been retrieved
  6. Return the median salary

```
CREATE PROCEDURE DEPT_MEDIAN
  (IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
  LANGUAGE SQL
  BEGIN
    DECLARE v_numRecords INTEGER DEFAULT 1;
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE c1 CURSOR FOR
      SELECT CAST(salary AS DOUBLE) FROM staff
        WHERE DEPT = deptNumber
        ORDER BY salary;
    DECLARE EXIT HANDLER FOR NOT FOUND
```

```
      SET medianSalary = 6666;
-- initialize OUT parameter
      SET medianSalary = 0;
      SELECT COUNT(*) INTO v_numRecords FROM staff
        WHERE DEPT = deptNumber;
      OPEN c1;
      WHILE v_counter < (v_numRecords / 2 + 1) DO
        FETCH c1 INTO medianSalary;
        SET v_counter = v_counter + 1;
      END WHILE;
      CLOSE c1;
  END
```

- *Example 2*: The following example illustrates the flow of execution in a hypothetical case where an UNDO handler is activated from another condition as the result of RESIGNAL:

```
CREATE PROCEDURE A()
LANGUAGE SQL
CS1: BEGIN ATOMIC
  DECLARE C CONDITION FOR SQLSTATE '12345';
  DECLARE D CONDITION FOR SQLSTATE '23456';

  DECLARE UNDO HANDLER FOR C
  H1: BEGIN
    -- Perform rollback after error, perform final cleanup, and exit
    -- procedure A.

    -- ...

    -- When this handler completes, execution continues after
    -- compound statement CS1; procedure A will terminate.
  END;

  -- Perform some work here ...
  CS2: BEGIN
    DECLARE CONTINUE HANDLER FOR D
    H2: BEGIN
      -- Perform local recovery, then forward the error
      -- condition to the outer handler for additional
      -- processing.

      -- ...

      RESIGNAL C; -- will activate UNDO handler H1; execution
                  -- WILL NOT return here.  Any local cursors
                  -- declared in H2 and CS2 will be closed.
    END;

    -- Perform some more work here ...

    -- Simulate raising of condition D by some SQL statement
    -- in compound statement CS2:
    SIGNAL D; -- will activate H2
  END;
END
```

CONNECT (type 1)

The CONNECT (Type 1) statement connects an application process to the identified application server according to the rules for remote unit of work.

An application process can only be connected to one application server at a time. This is called the *current server*. A default application server may be established when the application requester is initialized. If implicit connect is available and an application process is started, it is implicitly connected to the default application server. The application process can explicitly connect to a different application server by issuing a CONNECT statement. A connection lasts until a CONNECT RESET statement or a DISCONNECT statement is issued or until another CONNECT statement changes the application server.

Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared. When invoked using the command line processor, additional options can be specified. For more information, refer to "Using command line SQL statements and XQuery statements".

Authorization

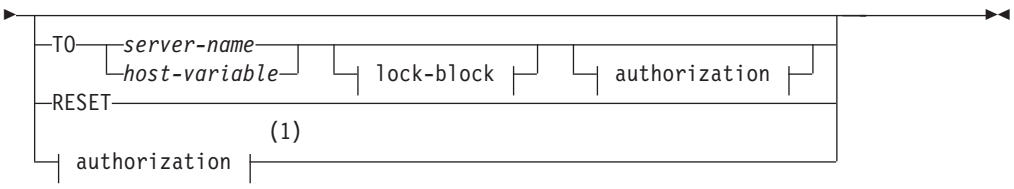
CONNECT processing goes through two levels of access control. Both levels must be satisfied for the connection to be successful.

The first level of access control is authentication, where the user ID associated with the connection must be successfully authenticated according to the authentication method set up for the server. At successful authentication, a DB2 authorization ID is derived from the connection user ID according to the authentication plug-in in effect for the server. This DB2 authorization ID must then pass the second level of access control for the connection, that is, authorization. To do so, this authorization ID must hold at least one of the following authorities:

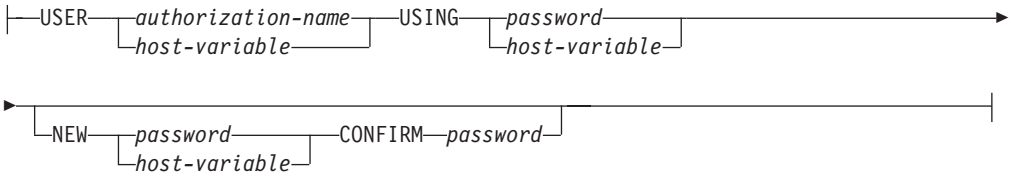
- CONNECT authority
- SECADM authority
- DBADM authority
- SYSADM authority
- SYSCTRL authority
- SYSMAINT authority
- SYSMON authority

Note: For a partitioned database, the user and group definitions must be identical across all database partitions.

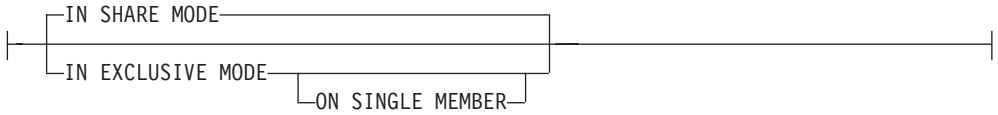
Syntax



authorization:



lock-block:



Notes:

1 This form is only valid if implicit connect is enabled.

Description

CONNECT (with no operand)

Returns information about the current server. The information is returned in the SQLERRP field of the SQLCA as described in "Successful Connection".

If a connection state exists, the authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If the authorization ID is longer than 8 bytes, it will be truncated to 8 bytes, and the truncation will be flagged in the SQLWARN0 and SQLWARN1 fields of the SQLCA, with 'W' and 'A', respectively.

If no connection exists and implicit connect is possible, then an attempt to make an implicit connection is made. If implicit connect is not available, this attempt results in an error (no existing connection). If no connection, then the SQLERRMC field is blank.

The territory code and code page of the application server are placed in the SQLERRMC field (as they are with a successful CONNECT statement).

This form of CONNECT:

- Does not require the application process to be in the connectable state.
- If connected, does not change the connection state.
- If unconnected and implicit connect is available, a connection to the default application server is made. In this case, the country or region code and code page of the application server are placed in the SQLERRMC field, like a successful CONNECT statement.
- If unconnected and implicit connect is not available, the application process remains unconnected.
- Does not close cursors.

TO *server-name* **or** *host-variable*
Identifies the application server by the specified *server-name* or a *host-variable* which contains the server-name.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-aligned and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

When the CONNECT statement is executed, the application process must be in the connectable state.

Successful Connection

If the CONNECT statement is successful:

- All open cursors are closed, all prepared statements are destroyed, and all locks are released from the previous application server.
- The application process is disconnected from its previous application server, if any, and connected to the identified application server.
- The actual name of the application server (not an alias) is placed in the CURRENT SERVER special register.
- Information about the application server is placed in the SQLERRP field of the SQLCA. If the application server is an IBM product, the information has the form *pppvvrrm*, where:
  - *ppp* identifies the product as follows:
    - `DSN` for DB2 for z/OS
    - `ARI` for DB2 Server for VSE & VM
    - `QSQ` for DB2 for i
    - `SQL` for DB2 for Linux, UNIX, and Windows
  - *vv* is a two-digit version identifier, such as '`08`'
  - *rr* is a two-digit release identifier, such as '`01`'
  - *m* is a one-character modification level identifier, such as '`0`'.

  For example, Version 9.5 of DB2 for Linux, UNIX, and Windows is identified as 'SQL09050'.
- The SQLERRMC field of the SQLCA is set to contain the following values (separated by X'FF')
  1. The country or region code of the application server (or blanks if using DB2 Connect),
  2. The code page of the application server (or CCSID if using DB2 Connect),
  3. The authorization ID (up to first 8 bytes only),
  4. The database alias,
  5. The platform type of the application server. Currently identified values are:

     Token Server

     QAS DB2 for i

     QDB2 DB2 for z/OS

     QDB2/6000
     DB2 Database for AIX

QDB2/HPUX
DB2 Database for HP-UX

QDB2/LINUX
DB2 Database for Linux

QDB2/NT
DB2 Database for Windows

QDB2/SUN
DB2 Database for Solaris Operating System

QSQLDS/VM
DB2 Server for VM

QSQLDS/VSE
DB2 Server for VSE

6. The agent ID. It identifies the agent executing within the database manager on behalf of the application. This field is the same as the **agent_id** element returned by the database monitor.
7. The agent index. It identifies the index of the agent and is used for service.
8. If the server instance operates in a DB2 pureScale environment, as indicated by SQLWARN0 and SQLWARN4 being set to 'W' and 'S' respectively, this value represents the member number. If, as indicated by token 10, the server instance operates in a partitioned environment, this token represents the node number. If the server instance operates in a non-partitioned environment and outside of a DB2 pureScale environment, this value is not applicable and is always 0.
9. The code page of the application client.
10. If this value is zero, the server instance operates in a non-partitioned environment and outside of a DB2 pureScale environment. Otherwise, this non-zero value represents the number of members in a DB2 pureScale instance, if SQLWARN0 and SQLWARN4 are set to 'W' and 'S' respectively. If this value is non-zero but neither SQLWARN0 nor SQLWARN4 is set, it represents the number of nodes in a partitioned environment. This token is present only with Version 5 or later.

- The SQLERRD(1) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
- The SQLERRD(2) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
- The SQLERRD(3) field of the SQLCA indicates whether or not the database on the connection is updatable. A database is initially updatable, but is changed to read-only if a unit of work determines the authorization ID cannot perform updates. The value is one of:
  – 1 - updatable

  - 2 - read-only
- The SQLERRD(4) field of the SQLCA returns certain characteristics of the connection. The value is one of:

  0 N/A (only possible if running from a client which is not at the latest level, is one-phase commit, and is an updater).

  1 one-phase commit.

  2 one-phase commit; read-only (only applicable to connections to DRDA1 databases in a TP Monitor environment).

  3 two-phase commit.
- The SQLERRD(5) field of the SQLCA returns the authentication type for the connection. The value is one of:

  0 Authenticated on the server.

  1 Authenticated on the client.

  2 Authenticated using DB2 Connect.

  4 Authenticated on the server with encryption.

  5 Authenticated using DB2 Connect with encryption.

  7 Authenticated using an external Kerberos security mechanism.

  9 Authenticated using an external GSS API plug-in security mechanism.

  11 Authenticated on the server, which accepts encrypted data.

  255 Authentication not specified.
- The SQLERRD(6) field of the SQLCA returns the database partition number of the database partition to which the connection was made if in a partitioned database environment. Otherwise, a value of 0 is returned.
- The SQLWARN1 field in the SQLCA will be set to 'A' if the authorization ID of the successful connection is longer than 8 bytes. This indicates that truncation has occurred. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.

Unsuccessful Connection

If the CONNECT statement is unsuccessful:

- The SQLERRP field of the SQLCA is set to the name of the module at the application requester that detected the error. The first three characters of the module name identify the product.
- If the CONNECT statement is unsuccessful because the application process is not in the connectable state, the connection state of the application process is unchanged.
- If the CONNECT statement is unsuccessful because the *server-name* is not listed in the local directory, an error message (SQLSTATE 08001) is issued and the connection state of the application process remains unchanged:
  - If the application requester was not connected to an application server then the application process remains unconnected.

  - If the application requester was already connected to an application server, the application process remains connected to that application server. Any further statements are executed at that application server.
- If the CONNECT statement is unsuccessful for any other reason, the application process is placed into the unconnected state.

IN SHARE MODE
Allows other concurrent connections to the database and prevents other users from connecting to the database in exclusive mode.

IN EXCLUSIVE MODE
Prevents concurrent application processes from executing any operations at the application server, unless they have the same authorization ID as the user holding the exclusive lock. This option is not supported by DB2 Connect.

ON SINGLE MEMBER
Specifies that the coordinator database member is connected in exclusive mode and all other members are connected in share mode. If the database is neither in a partitioned environment nor a DB2 pureScale environment, this option can be specified, but it has no effect.

RESET
Disconnects the application process from the current server. A commit operation is performed. If implicit connect is available, the application process remains unconnected until an SQL statement is issued.

USER *authorization-name/host-variable*
Identifies the user ID trying to connect to the application server. If a host-variable is specified, it must be a character string variable that does not include an indicator variable. The user ID that is contained within the *host-variable* must be left-aligned and must not be delimited by quotation marks.

USING *password/host-variable*
Identifies the password of the user ID trying to connect to the application server. The *password* or *host-variable* can be up to 14 bytes long. If a host variable is specified, it must be a character string variable with a length attribute not greater than 14, and it must not include an indicator variable.

NEW *password/host-variable* **CONFIRM** *password*
Identifies the new password that should be assigned to the user ID identified by the USER option. The *password* or *host-variable* can be up to 14 bytes long. If a host variable is specified, it must be a character string variable with a length attribute not greater than 14, and it must not include an indicator variable. The system on which the password will be changed depends on how the user authentication has been set up. New passwords can be assigned using this clause on the following servers for the indicated (and later) releases: DB2 Universal Database™ Version 8 on AIX and Windows operating systems, DB2 Version 9.1 Fix Pack 3 or later on Linux operating systems, DB2 for z/OS Version 7, DB2 for i5/OS V6R1. To support the changing passwords for DB2 database products on Linux, the DB2 instance must be configured to use the security plug-ins IBMOSchgpwdclient and IBMOSchgpwdserver.

Notes

- It is good practice for the first SQL statement executed by an application process to be the CONNECT statement.
- If a CONNECT statement is issued to the current application server with a different user ID and password then the conversation is deallocated and

reallocated. All cursors are closed by the database manager (with the loss of the cursor position if the WITH HOLD option was used).

- If a CONNECT statement is issued to the current application server with the same user ID and password then the conversation is not deallocated and reallocated. Cursors, in this case, are not closed.
- To use a multiple-partition partitioned database environment, the user or application must connect to one of the database partitions listed in the `db2nodes.cfg` file. You should try to ensure that not all users use the same database partition as the coordinator partition.
- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft Windows Security Account Manager (SAM)-compatible name.
- The database can be inaccessible if the database was not explicitly activated, a client application performs frequent reconnections, or the time interval between issuing the **DEACTIVATE DATABASE** and **ACTIVATE DATABASE** commands is very short. Activate the database by issuing the **ACTIVATE DATABASE** command and then attempt to connect to the database.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DBPARTITIONNUM or NODE can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

Examples

- *Example 1:* In a C program, connect to the application server TOROLAB, using database alias TOROLAB, user ID FERMAT, and password THEOREM.

```
EXEC SQL  CONNECT TO TOROLAB USER FERMAT USING THEOREM;
```

- *Example 2:* In a C program, connect to an application server whose database alias is stored in the host variable APP_SERVER (varchar(8)). Following a successful connection, copy the 3-character product identifier of the application server to the variable PRODUCT (char(3)).

```
EXEC SQL  CONNECT TO :APP_SERVER;
if (strncmp(SQLSTATE,'00000',5))
  strncpy(PRODUCT,sqlca.sqlerrp,3);
```

CONNECT (type 2)

The CONNECT (Type 2) statement connects an application process to the identified application server and establishes the rules for application-directed distributed unit of work. This server is then the current server for the process.

Most aspects of a CONNECT (Type 1) statement also apply to a CONNECT (Type 2) statement. Rather than repeating that material here, this section describes only those elements of Type 2 that differ from Type 1.

Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared. When invoked using the command line processor, additional options can be specified. For more information, refer to "Using command line SQL statements and XQuery statements".

Authorization

CONNECT processing goes through two levels of access control. Both levels must be satisfied for the connection to be successful.

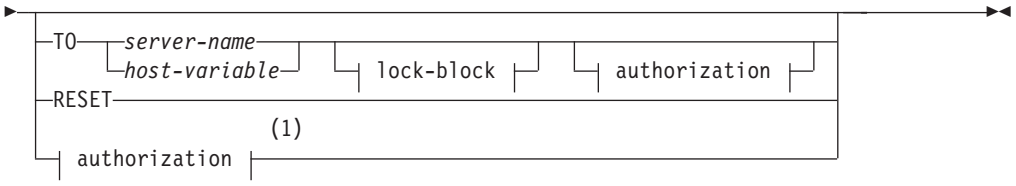
The first level of access control is authentication, where the user ID associated with the connection must be successfully authenticated according to the authentication method set up for the server. At successful authentication, a DB2 authorization ID is derived from the connection user ID according to the authentication plug-in in effect for the server. This DB2 authorization ID must then pass the second level of access control for the connection, that is, authorization. To do so, this authorization ID must hold at least one of the following authorities:

- CONNECT authority
- SECADM authority
- DBADM authority
- SYSADM authority
- SYSCTRL authority
- SYSMAINT authority
- SYSMON authority

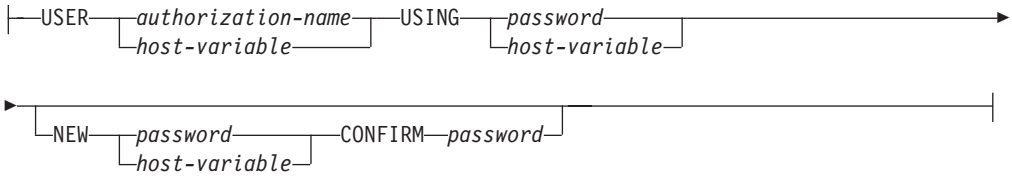
Note: For a partitioned database, the user and group definitions must be identical across all database partitions.

Syntax

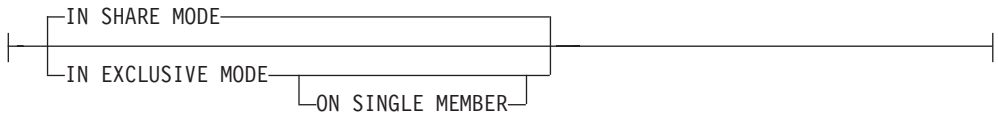
The selection between Type 1 and Type 2 is determined by precompiler options. For an overview of these options, see "Connecting to distributed relational databases".



authorization:



lock-block:



Notes:

1 This form is only valid if implicit connect is enabled.

Description

TO *server-name/host-variable*

The rules for coding the name of the server are the same as for Type 1.

If the SQLRULES(STD) option is in effect, the *server-name* must not identify an existing connection of the application process, otherwise an error (SQLSTATE 08002) is raised.

If the SQLRULES(DB2) option is in effect and the *server-name* identifies an existing connection of the application process, that connection is made current and the old connection is placed into the dormant state. That is, the effect of the CONNECT statement in this situation is the same as that of a SET CONNECTION statement.

For information about the specification of SQLRULES, see "Options that Govern Distributed Unit of Work Semantics".

Successful Connection

If the CONNECT statement is successful:

- A connection to the application server is either created (or made non-dormant) and placed into the current and held states.
- If the CONNECT TO is directed to a different server than the current server, then the current connection is placed into the dormant state.
- The CURRENT SERVER special register and the SQLCA are updated in the same way as for CONNECT (Type 1).

Unsuccessful Connection

If the CONNECT statement is unsuccessful:

- No matter what the reason for failure, the connection state of the application process and the states of its connections are unchanged.
- As with an unsuccessful Type 1 CONNECT, the SQLERRP field of the SQLCA is set to the name of the module at the application requester or server that detected the error.

CONNECT (with no operand), IN SHARE/EXCLUSIVE MODE, USER, and USING
If a connection exists, Type 2 behaves like a Type 1. The authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If a connection does not exist, no attempt to make an implicit connection is made and the SQLERRP and SQLERRMC fields return a blank. (Applications can check if a current connection exists by checking these fields.)

A CONNECT with no operand that includes USER and USING can still connect an application process to a database using the DB2DBDFT environment variable. This method is equivalent to a Type 2 CONNECT RESET, but permits the use of a user ID and password.

RESET
Equivalent to an explicit connect to the default database if it is available. If a default database is not available, the connection state of the application process and the states of its connections are unchanged.

Availability of a default database is determined by installation options, environment variables, and authentication settings.

Rules

- As outlined in "Options that Govern Distributed Unit of Work Semantics", a set of connection options governs the semantics of connection management. Default values are assigned to every preprocessed source file. An application can consist of multiple source files precompiled with different connection options.

  Unless a SET CLIENT command or API has been executed first, the connection options used when preprocessing the source file containing the first SQL statement executed at run time become the effective connection options.

  If a CONNECT statement from a source file preprocessed with different connection options is subsequently executed without the execution of any intervening SET CLIENT command or API, an error (SQLSTATE 08001) is returned. Note that once a SET CLIENT command or API has been executed, the connection options used when preprocessing all source files in the application are ignored.

  Example 1 in the "Examples" section of this statement illustrates these rules.
- Although the CONNECT statement can be used to establish or switch connections, CONNECT with the USER/USING clause will only be accepted when there is no current or dormant connection to the named server. The connection must be released before issuing a connection to the same server with the USER/USING clause, otherwise it will be rejected (SQLSTATE 51022). Release the connection by issuing a DISCONNECT statement or a RELEASE statement followed by a COMMIT statement.

Comparing Type 1 and Type 2 CONNECT Statements

The semantics of the CONNECT statement are determined by the CONNECT precompiler option or the SET CLIENT API (see "Options that Govern Distributed Unit of Work Semantics"). CONNECT Type 1 or CONNECT Type 2 can be

specified and the CONNECT statements in those programs are known as Type 1 and Type 2 CONNECT statements, respectively. Their semantics are described in the following tables:

Use of **CONNECT**:

Type 1	Type 2
Each unit of work can only establish connection to one application server.	Each unit of work can establish connection to multiple application servers.
The current unit of work must be committed or rolled back before allowing a connection to another application server.	The current unit of work need not be committed or rolled back before connecting to another application server.
The CONNECT statement establishes the current connection. Subsequent SQL requests are forwarded to this connection until changed by another CONNECT.	Same as Type 1 CONNECT if establishing the first connection. If switching to a dormant connection and SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to the current connection is valid and does not change the current connection.	Same as Type 1 CONNECT if the SQLRULES precompiler option is set to DB2. If SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to another application server disconnects the current connection. The new connection becomes the current connection. Only one connection is maintained in a unit of work.	Connecting to another application server puts the current connection into the *dormant state*. The new connection becomes the current connection. Multiple connections can be maintained in a unit of work. If the CONNECT is for an application server on a dormant connection, it becomes the current connection. Connecting to a dormant connection using CONNECT is only allowed if SQLRULES(DB2) was specified. If SQLRULES(STD) was specified, then the SET CONNECTION statement must be used instead.
SET CONNECTION statement is supported for Type 1 connections, but the only valid target is the current connection.	SET CONNECTION statement is supported for Type 2 connections to change the state of a connection from dormant to current.

Use of **CONNECT...USER...USING**:

Type 1	Type 2
Connecting with the USER...USING clauses disconnects the current connection and establishes a new connection with the given authorization name and password.	Connecting with the USER/USING clause will only be accepted when there is no current or dormant connection to the same named server.

Use of **Implicit CONNECT, CONNECT RESET**, and **Disconnecting**:

Type 1	Type 2
CONNECT RESET can be used to disconnect the current connection.	CONNECT RESET is equivalent to connecting to the default application server explicitly if one has been defined in the system. Connections can be disconnected by the application at a successful COMMIT. Prior to the commit, use the RELEASE statement to mark a connection as release-pending. All such connections will be disconnected at the next COMMIT. An alternative is to use the precompiler options DISCONNECT(EXPLICIT), DISCONNECT(CONDITIONAL), DISCONNECT(AUTOMATIC), or the DISCONNECT statement instead of the RELEASE statement.
After using CONNECT RESET to disconnect the current connection, if the next SQL statement is not a CONNECT statement, then it will perform an implicit connect to the default application server if one has been defined in the system.	CONNECT RESET is equivalent to an explicit connect to the default application server if one has been defined in the system.
It is an error to issue consecutive CONNECT RESETs.	It is an error to issue consecutive CONNECT RESETs ONLY if SQLRULES(STD) was specified because this option disallows the use of CONNECT to existing connection.
CONNECT RESET implicitly rolls back the current unit of work.	CONNECT RESET implicitly rolls back the current unit of work.
If an existing connection is disconnected by the system for whatever reasons, then subsequent non-CONNECT SQL statements to this database will receive an SQLSTATE of 08003.	If an existing connection is disconnected by the system, COMMIT, ROLLBACK, and SET CONNECTION statements are still permitted.
The unit of work will be implicitly committed when the application process terminates successfully.	Same as Type 1.
All connections (only one) are disconnected when the application process terminates.	All connections (current, dormant, and those marked for release pending) are disconnected when the application process terminates.

CONNECT Failures:

Type 1	Type 2
Regardless of whether there is a current connection when a CONNECT fails (with an error other than server-name not defined in the local directory), the application process is placed in the unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.	If there is a current connection when a CONNECT fails, the current connection is unaffected. If there was no current connection when the CONNECT fails, then the program is then in an unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.

Notes

- Implicit connect is supported for the first SQL statement in an application with Type 2 connections. In order to execute SQL statements on the default database, first the CONNECT RESET or the CONNECT USER/USING statement must be used to establish the connection. The CONNECT statement with no operands will display information about the current connection if there is one, but will not connect to the default database if there is no current connection.
- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft Windows Security Account Manager (SAM)-compatible name.
- ***Termination of a connection***: When a connection is terminated and a transaction has not yet been committed or rolled back, see "Use of Implicit CONNECT, CONNECT RESET, and Disconnecting" section for details on what happens to such transactions. To ensure consistent behavior, code an explicit COMMIT statement or ROLLBACK statement instead of depending on the behavior of the CONNECT statement.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DBPARTITIONNUM or NODE can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

Examples

- *Example 1:* This example illustrates the use of multiple source programs (shown in the boxes), some preprocessed with different connection options (shown in the statement preceding the code), and one of which contains a SET CLIENT API call.

  PGM1: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO OTTAWA;
exec sql SELECT col1 INTO :hv1
FROM tbl1;
...
```

  PGM2: CONNECT(2) SQLRULES(STD) DISCONNECT(AUTOMATIC)

```
...
exec sql CONNECT TO QUEBEC;
exec sql SELECT col1 INTO :hv1
FROM tbl2;
...
```

  PGM3: CONNECT(2) SQLRULES(STD) DISCONNECT(EXPLICIT)

```
...
SET CLIENT CONNECT 2  SQLRULES DB2  DISCONNECT EXPLICIT [1]
exec sql CONNECT TO LONDON;
exec sql SELECT col1 INTO :hv1
FROM tbl3;
...
```

  Note:

  1. Not the actual syntax of the SET CLIENT API

  PGM4: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO REGINA;
exec sql SELECT col1 INTO :hv1
FROM tbl4;
...
```

If the application executes PGM1 then PGM2:

– connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

– connect to QUEBEC fails with SQLSTATE 08001 because both SQLRULES and DISCONNECT are different.

If the application executes PGM1 then PGM3:

– connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

– connect to LONDON runs: connect=2, sqlrules=DB2, disconnect=EXPLICIT

This is OK because the SET CLIENT API is run before the second CONNECT statement.

If the application executes PGM1 then PGM4:

– connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

– connect to REGINA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

This is OK because the preprocessor options for PGM1 are the same as those for PGM4.

- *Example 2:* This example shows the interrelationships of the CONNECT (Type 2), SET CONNECTION, RELEASE, and DISCONNECT statements. S0, S1, S2, and S3 represent four servers.

Sequence	Statement	Current Server	Dormant Connections	Release Pending
0	• No statement	• None	• None	• None
1	• SELECT * FROM TBLA	• S0 (default)	• None	• None
2	• CONNECT TO S1	• S1	• S0	• None
	• SELECT * FROM TBLB	• S1	• S0	• None
3	• CONNECT TO S2	• S2	• S0, S1	• None
	• UPDATE TBLC SET ...	• S2	• S0, S1	• None
4	• CONNECT TO S3	• S3	• S0, S1, S2	• None
	• SELECT * FROM TBLD	• S3	• S0, S1, S2	• None
5	• SET CONNECTION S2	• S2	• S0, S1, S3	• None
6	• RELEASE S3	• S2	• S0, S1	• S3
7	• COMMIT	• S2	• S0, S1	• None
8	• SELECT * FROM TBLE	• S2	• S0, S1	• None
9	• DISCONNECT S1	• S2	• S0	• None
	• SELECT * FROM TBLF	• S2	• S0	• None

CREATE ALIAS

The CREATE ALIAS statement defines an alias for a module, nickname, sequence, table, view, or another alias. Aliases are also known as synonyms.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the alias does not exist
- CREATEIN privilege on the schema, if the schema name of the alias refers to an existing schema, or CREATEIN privilege on SYSPUBLIC, if a public alias is being created
- DBADM authority

Privileges required to use the referenced object through its alias are identical to the privileges required to use the object directly.

To replace an existing alias, the authorization ID of the statement must be the owner of the existing alias (SQLSTATE 42501).

Syntax

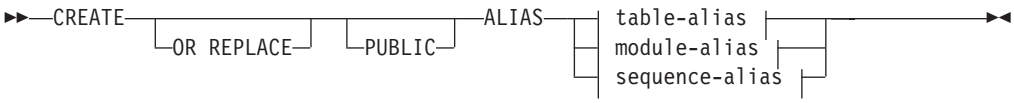
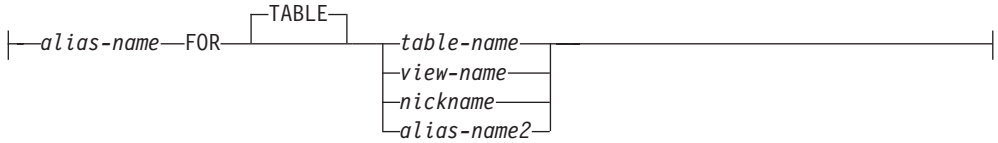
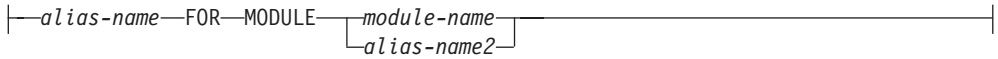


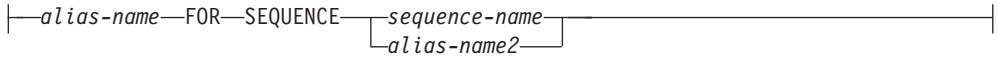
table-alias:



module-alias:



sequence-alias:



Description

OR REPLACE
Specifies to replace the definition for the alias if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog. This option is ignored if a definition for the alias does not exist at the current server. This option can be specified only by the owner of the object.

PUBLIC
Specifies that the alias is an object in the system schema SYSPUBLIC.

alias-name
Names the alias. For a table alias, the name must not identify a nickname, table, view, or table alias that exists at the current server. For a module alias, the name must not identify a module or module alias that exists at the current server. For a sequence alias, the name must not identify a sequence or sequence alias that exists at the current server.

If a two-part name is specified, the schema name cannot begin with 'SYS' (SQLSTATE 42939) except if PUBLIC is specified, then the schema name must be SYSPUBLIC (SQLSTATE 428EK).

FOR TABLE *table-name*, *view-name*, *nickname*, **or** *alias-name2*
Identifies the table, view, nickname, or table alias for which *alias-name* is defined. If another alias name is supplied (*alias-name2*), then it must not be the same as the new *alias-name* being defined (in its fully-qualified form). The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

FOR MODULE *module-name*, **or** *alias-name2*
Identifies the module or module alias for which *alias-name* is defined. If another alias name is supplied (*alias-name2*), then it must not be the same as the new alias-name being defined (in its fully-qualified form).

FOR SEQUENCE *sequence-name*, **or** *alias-name2*
Identifies the sequence or sequence alias for which *alias-name* is defined. If another alias name is supplied (*alias-name2*), then it must not be the same as the new *alias-name* being defined (in its fully-qualified form). The *sequence-name* must not be a sequence generated by the system for an identity column (SQLSTATE 428FB).

Notes

- The keyword PUBLIC is used to create a public alias (also known as a public synonym). If the keyword PUBLIC is not used, the type of alias is a private alias (also known as a private synonym).
- The definition of the newly created table alias is stored in SYSCAT.TABLES. The definition of the newly created module alias is stored in SYSCAT.MODULES. The definition of the newly created sequence alias is stored in SYSCAT.SEQUENCES.
- An alias can be defined for an object that does not exist at the time of the definition. If it does not exist, a warning is issued (SQLSTATE 01522). However, the referenced object must exist when a SQL statement containing the alias is compiled, otherwise an error is issued (SQLSTATE 52004).
- An alias can be defined to refer to another alias as part of an alias chain but this chain is subject to the same restrictions as a single alias when used in an SQL statement. An alias chain is resolved in the same way as a single alias. If an alias used in a statement in a package, an SQL routine, a trigger, the default expression for a global variable, or a view definition points to an alias chain,

then a dependency is recorded for the package, SQL routine, trigger, global variable, or view on each alias in the chain. An alias cannot refer to itself in an alias chain and such a cycle is detected at alias definition time (SQLSTATE 42916).

- **Resolving an unqualified alias name:** When resolving an unqualified name, private aliases are considered before public aliases.
- **Conservative binding for public aliases:** If a public alias is used in a statement in a package, an SQL routine, a trigger, the default expression for a global variable, or a view definition, the public alias will continue to be used by these objects regardless of what other object with the same name is created subsequently.
- Creating an alias with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products.
  - SYNONYM can be specified in place of ALIAS

Examples

- *Example 1:* HEDGES attempts to create an alias for a table T1 (both unqualified).

```
CREATE ALIAS A1 FOR T1
```

  The alias HEDGES.A1 is created for HEDGES.T1.

- *Example 2:* HEDGES attempts to create an alias for a table (both qualified).

```
CREATE ALIAS HEDGES.A1 FOR MCKNIGHT.T1
```

  The alias HEDGES.A1 is created for MCKNIGHT.T1.

- *Example 3:* HEDGES attempts to create an alias for a table (alias in a different schema; HEDGES is not a DBADM; HEDGES does not have CREATEIN on schema MCKNIGHT).

```
CREATE ALIAS  MCKNIGHT.A1 FOR MCKNIGHT.T1
```

  This example fails (SQLSTATE 42501).

- *Example 4:* HEDGES attempts to create an alias for an undefined table (both qualified; FUZZY.WUZZY does not exist).

```
CREATE ALIAS HEDGES.A1 FOR FUZZY.WUZZY
```

  This statement succeeds but with a warning (SQLSTATE 01522).

- *Example 5:* HEDGES attempts to create an alias for an alias (both qualified).

```
CREATE ALIAS HEDGES.A1 FOR MCKNIGHT.T1
CREATE ALIAS HEDGES.A2 FOR HEDGES.A1
```

  The first statement succeeds (as per example 2).

  The second statement succeeds and an alias chain is created, consisting of HEDGES.A2 which refers to HEDGES.A1 which refers to MCKNIGHT.T1. Note that it does not matter whether or not HEDGES has any privileges on MCKNIGHT.T1. The alias is created regardless of the table privileges.

- *Example 6:* Designate A1 as an alias for the nickname FUZZYBEAR.

```
CREATE ALIAS A1 FOR FUZZYBEAR
```

- *Example 7:* A large organization has a finance department numbered D108 and a personnel department numbered D577. D108 keeps certain information in a table that resides at a DB2 RDBMS. D577 keeps certain records in a table that resides at an Oracle RDBMS. A DBA defines the two RDBMSs as data sources within a federated system, and gives the tables the nicknames of DEPTD108 and DEPTD577, respectively. A federated system user needs to create joins between these tables, but would like to reference them by names that are more meaningful than their alphanumeric nicknames. So the user defines FINANCE as an alias for DEPTD108 and PERSONNEL as an alias for DEPTD577.

  ```
  CREATE ALIAS FINANCE FOR DEPTD108
  CREATE ALIAS PERSONNEL FOR DEPTD577
  ```

- *Example 8:* Create a public alias called TABS for the catalog view SYSCAT.TABLES.

  ```
  CREATE PUBLIC ALIAS TABS FOR SYSCAT.TABLES
  ```

CREATE AUDIT POLICY

The CREATE AUDIT POLICY statement defines an auditing policy at the current server. The policy determines what categories are to be audited; it can then be applied to other database objects to determine how the use of those objects is to be audited.

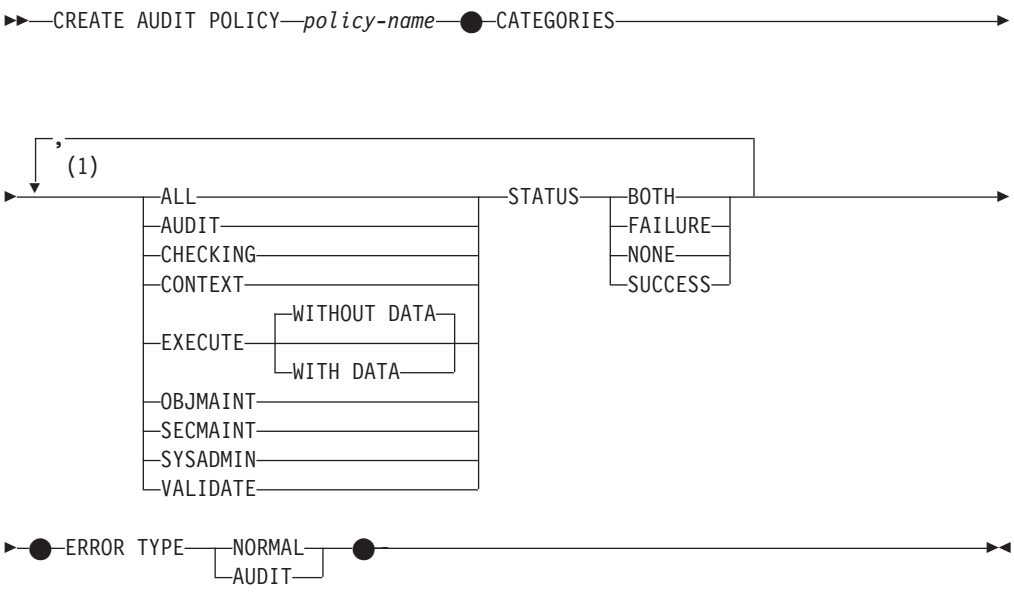
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Notes:

1 Each category can be specified at most once (SQLSTATE 42614), and no other category can be specified if ALL is specified (SQLSTATE 42601).

Description

policy-name
Names the audit policy. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *policy-name* must not identify an audit policy already described in the catalog (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

CATEGORIES
A list of one or more audit categories for which a status is specified. If ALL is not specified, the STATUS of any category that is not explicitly specified is set to NONE.

ALL
Sets all categories to the same status. The EXECUTE category is WITHOUT DATA.

AUDIT
Generates records when audit settings are changed or when the audit log is accessed.

CHECKING
Generates records during authorization checking of attempts to access or manipulate database objects or functions.

CONTEXT
Generates records to show the operation context when a database operation is performed.

EXECUTE
Generates records to show the execution of SQL statements.

WITHOUT DATA or WITH DATA
Specifies whether or not input data values provided for any host variables and parameter markers should be logged as part of the EXECUTE category.

WITHOUT DATA
Input data values provided for any host variables and parameter markers are not logged as part of the EXECUTE category. WITHOUT DATA is the default.

WITH DATA
Input data values provided for any host variables and parameter markers are logged as part of the EXECUTE category. Not all input values are logged; specifically, LOB, LONG, XML, and structured type parameters appear as the null value. Date, time, and timestamp fields are logged in ISO format. The input data values are converted to the database code page before being logged. If code page conversion fails, no errors are returned and the unconverted data is logged.

OBJMAINT
Generates records when data objects are created or dropped.

SECMAINT
Generates records when object privileges, database privileges, or DBADM authority is granted or revoked. Records are also generated when the database manager security configuration parameters **sysadm_group**, **sysctrl_group**, or **sysmaint_group** are modified.

SYSADMIN
Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.

VALIDATE
Generates records when users are authenticated or when system security information related to a user is retrieved.

STATUS
Specifies a status for the specified category.

BOTH
Successful and failing events will be audited.

FAILURE
Only failing events will be audited.

SUCCESS
Only successful events will be audited.

NONE
No events in this category will be audited.

ERROR TYPE
Specifies whether audit errors are to be returned or ignored.

NORMAL
Any errors generated by the audit are ignored and only the SQLCODEs for errors associated with the operation being performed are returned to the application.

AUDIT
All errors, including errors occurring within the audit facility itself, are returned to the application.

Rules

- An AUDIT-exclusive SQL statement must be followed by a COMMIT or ROLLBACK statement (SQLSTATE 5U021). AUDIT-exclusive SQL statements are:
  - AUDIT
  - CREATE AUDIT POLICY, ALTER AUDIT POLICY, or DROP (AUDIT POLICY)
  - DROP (ROLE or TRUSTED CONTEXT if it is associated with an audit policy)
- An AUDIT-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Only one uncommitted AUDIT-exclusive SQL statement is allowed at a time across all database partitions. If an uncommitted AUDIT-exclusive SQL statement is executing, subsequent AUDIT-exclusive SQL statements wait until the current AUDIT-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.

Example

Create an audit policy to audit successes and failures for the AUDIT and OBJMAINT categories; only failures for the SECMAINT, CHECKING, and VALIDATE categories, and no events for the other categories.

```
CREATE AUDIT POLICY DBAUDPRF
  CATEGORIES AUDIT    STATUS BOTH,
             SECMAINT STATUS FAILURE,
             OBJMAINT STATUS BOTH,
             CHECKING STATUS FAILURE,
             VALIDATE STATUS FAILURE
  ERROR TYPE NORMAL
```

CREATE BUFFERPOOL

The CREATE BUFFERPOOL statement defines a buffer pool at the current server. Buffer pools are defined on members which can access data partitions.

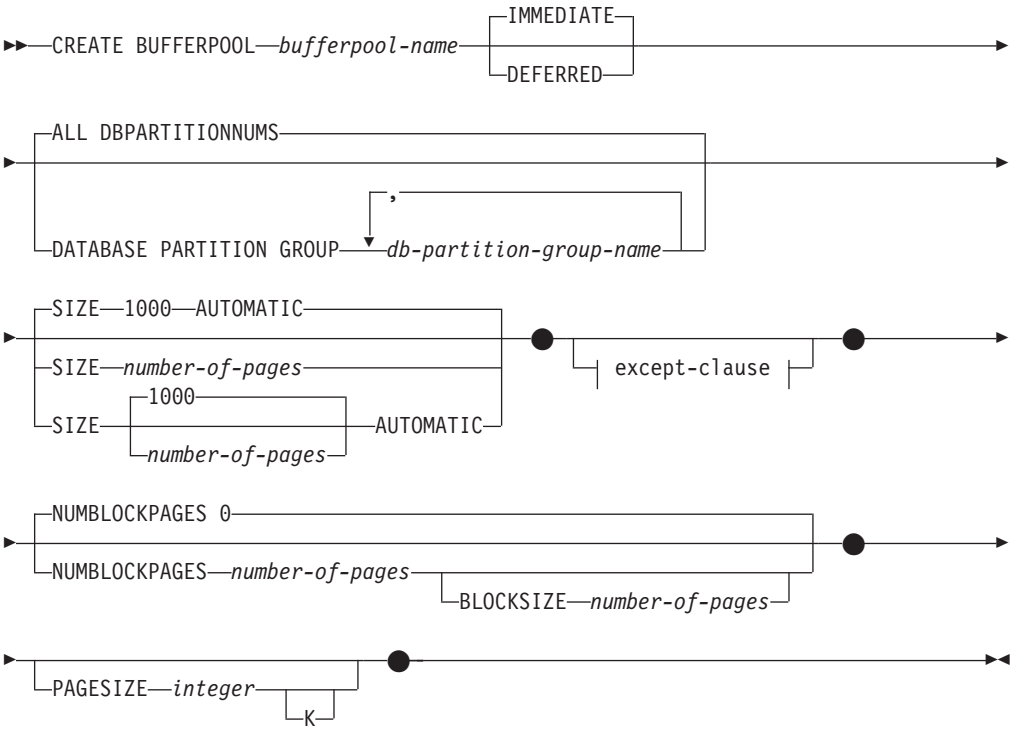
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

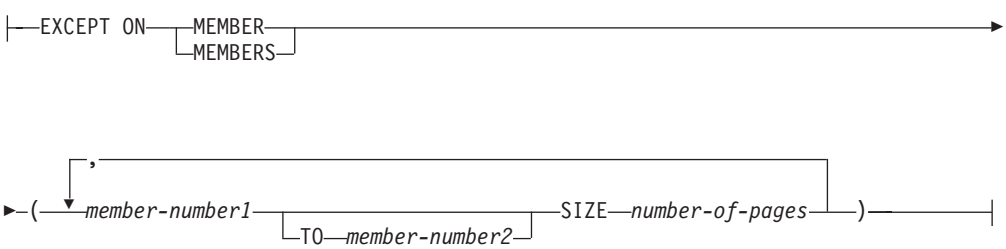
Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax



except-clause:



Description

bufferpool-name
Names the buffer pool. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *bufferpool-name* must not identify a buffer pool that already exists in the catalog (SQLSTATE 42710). The *bufferpool-name* must not begin with the characters 'SYS' (SQLSTATE 42939).

IMMEDIATE or DEFERRED
Indicates whether or not the buffer pool will be created immediately.

IMMEDIATE
The buffer pool will be created immediately. If there is not enough reserved space in the database shared memory to allocate the new buffer pool (SQLSTATE 01657) the statement is executed as DEFERRED.

DEFERRED
The buffer pool will be created when the database is deactivated (all applications need to be disconnected from the database). Reserved memory space is not needed; DB2 will allocate the required memory from the system.

ALL DBPARTITIONNUMS or DATABASE PARTITION GROUP
Identifies the members on which the buffer pool is to be defined. The default is ALL DBPARTITIONNUMS.

ALL DBPARTITIONNUMS
This buffer pool will be created on all members which can access all data partitions in the database.

DATABASE PARTITION GROUP *db-partition-group-name, ...*
Identifies the database partition group or groups to which the buffer pool definition applies. The buffer pool will be created only on members in the specified database partition groups. Each database partition group must exist in the database (SQLSTATE 42704).

SIZE
Specifies the size of the buffer pool. This size will be the default size for all members on which the buffer pool exists. The default is 1000 pages.

number-of-pages
The number of pages for the new buffer pool. The minimum number of pages is 2 and the maximum is architecture-dependent (SQLSTATE 42615).

AUTOMATIC
Enables self tuning for this buffer pool. The database manager adjusts the size of the buffer pool in response to workload requirements. The implicit or explicit number of pages specified is used as the initial size of the buffer pool. On subsequent database activations the buffer pool size is based on the last tuning value determined by the self-tuning memory manager (STMM). Note that STMM enforces a minimum size for automatic buffer pools. To determine the current size of buffer pools that are enabled for self tuning, use the **GET SNAPSHOT** command and examine the current size of the buffer pools (the value of the **bp_cur_buffsz** monitor element).

NUMBLOCKPAGES *number-of-pages*
Specifies the number of pages that should exist in the block-based area. The number of pages must not be greater than 98 percent of the number of pages for the buffer pool (SQLSTATE 54052). Specifying the value 0 disables block I/O. The actual value of NUMBLOCKPAGES used will be a multiple of BLOCKSIZE.

NUMBLOCKPAGES is not supported in a DB2 pureScale environment (SQLSTATE 56038).

BLOCKSIZE *number-of-pages*
Specifies the number of pages in a block. The block size must be a value between 2 and 256 (SQLSTATE 54053). The default value is 32.

BLOCKSIZE is not supported in a DB2 pureScale environment (SQLSTATE 56038).

EXCEPT ON MEMBER or EXCEPT ON MEMBERS
Specifies the member or members for which the size of the buffer pool will be different than the default specified for the database partition group to which the member has access. If this clause is not specified, all members that can access the data partitions in the specified database partition group will have the same size as specified for this buffer pool.

member-number1
Specifies a member number for a member that has access to a data partition for which the buffer pool is created (SQLSTATE 42729).

TO *member-number2*
Specifies a range of member numbers. The value of *member-number2* must be greater than or equal to the value of *member-number1* (SQLSTATE 428A9). Each member identified by the member number range inclusive must have access to the data partition for which the buffer pool is created (SQLSTATE 428A9).

SIZE *number-of-pages*
The size of the buffer pool specified as the number of pages. The minimum number of pages is 2 and the maximum is architecture-dependent (SQLSTATE 42615).

PAGESIZE *integer* **[K]**
Defines the size of pages used for the buffer pool. The valid values for *integer* without the suffix K are 4096, 8192, 16 384, or 32 768. The valid values for *integer* with the suffix K are 4, 8, 16, or 32. Any number of spaces is allowed between *integer* and K, including no space. If the page size is not one of these values, an error is returned (SQLSTATE 428DE).

The default value is provided by the **pagesize** database configuration parameter, which is set when the database is created.

Notes

- If the buffer pool is created using the DEFERRED option, any table space created in this buffer pool will use a small system buffer pool of the same page size, until next database activation. The database has to be restarted for the buffer pool to become active and for table space assignments to the new buffer pool to take effect. The default option is IMMEDIATE.
- There should be enough real memory on the machine for the total of all the buffer pools, as well as for the rest of the database manager and application requirements. If DB2 is unable to obtain memory for the regular buffer pools, it will attempt to start with small system buffer pools, one for each page size (4K, 8K, 16K and 32K). In this situation, a warning will be returned to the user (SQLSTATE 01626), and the pages from all table spaces will use the system buffer pools.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.

- NODEGROUP can be specified in place of DATABASE PARTITION GROUP
- DBPARTITIONNUM or NODE can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON
- DBPARTITIONNUMS or NODES can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON

CREATE DATABASE PARTITION GROUP

The CREATE DATABASE PARTITION GROUP statement defines a new database partition group within the database, assigns database partitions to the database partition group, and records the database partition group definition in the system catalog.

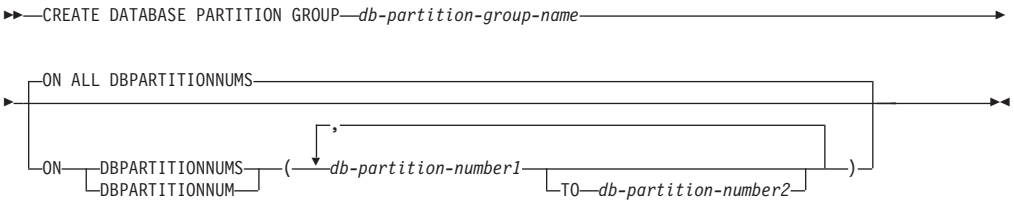
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax



Description

db-partition-group-name
Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *db-partition-group-name* must not identify a database partition group that already exists in the catalog (SQLSTATE 42710). The *db-partition-group-name* must not begin with the characters 'SYS' or 'IBM' (SQLSTATE 42939).

ON ALL DBPARTITIONNUMS
Specifies that the database partition group is defined over all database partitions defined to the database (`db2nodes.cfg` file) at the time the database partition group is created.

If a database partition is added to the database system, the ALTER DATABASE PARTITION GROUP statement should be issued to include this new database partition in a database partition group (including IBMDEFAULTGROUP). Furthermore, the REDISTRIBUTE DATABASE PARTITION GROUP command must be issued to move data to the database partition.

ON DBPARTITIONNUMS
Specifies the database partitions that are in the database partition group. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1
Specify a database partition number. (A *node-name* of the form `NODE`*nnnnn* can be specified for compatibility with the previous version.)

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of

db-partition-number1 (SQLSTATE 428A9). All database partitions between and including the specified database partition numbers are included in the database partition group.

Rules

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).
- Each *db-partition-number* listed in the ON DBPARTITIONNUMS clause must be appear at most once (SQLSTATE 42728).
- A valid *db-partition-number* is between 0 and 999 inclusive (SQLSTATE 42729).
- The CREATE DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Notes

- This statement creates a distribution map for the database partition group. A distribution map identifier (PMAP_ID) is generated for each distribution map. This information is recorded in the catalog and can be retrieved from SYSCAT.DBPARTITIONGROUPS and SYSCAT.PARTITIONMAPS. Each entry in the distribution map specifies the target database partition on which all rows that are hashed reside. For a single-partition database partition group, the corresponding distribution map has only one entry. For a multiple partition database partition group, the corresponding distribution map has 32768 entries, where the database partition numbers are assigned to the map entries in a round-robin fashion, by default.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NODE can be specified in place of DBPARTITIONNUM
  - NODES can be specified in place of DBPARTITIONNUMS
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples

The following examples are based on a partitioned database with six database partitions defined as 0, 1, 2, 5, 7, and 8.

- *Example 1*: Assume that you want to create a database partition group called MAXGROUP on all six database partitions. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MAXGROUP ON ALL DBPARTITIONNUMS
```

- *Example 2*: Assume that you want to create a database partition group called MEDGROUP on database partitions 0, 1, 2, 5, and 8. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MEDGROUP
  ON DBPARTITIONNUMS( 0 TO 2, 5, 8)
```

- *Example 3*: Assume that you want to create a single-partition database partition group MINGROUP on database partition 7. The statement is as follows:

```
CREATE DATABASE PARTITION GROUP MINGROUP
  ON DBPARTITIONNUM (7)
```

CREATE EVENT MONITOR

The CREATE EVENT MONITOR statement defines a monitor that will record certain events that occur when using the database. The definition of each event monitor also specifies where the database should record the events.

Several different types of event monitors can be created using this statement. Seven of these types are described separately (see Related links) and the remaining types are described here. The types of event monitors described separately are:

- Activities. The event monitor will record activity events that occur when using the database. The definition of the statistics event monitor also specifies where the database should record the events.
- Locking. The event monitor will record lock-related events that occur when using the database. All records are collected in the unformatted event table.
- Package cache. The event monitor will record events related to the package cache statement.
- Statistics. The event monitor will record statistics events that occur when using the database. The definition of the statistics event monitor also specifies where the database should record the events.
- Threshold violations. The event monitor will record threshold violation events that occur when using the database. The definition of the statistics event monitor also specifies where the database should record the events.
- Unit of work. The event monitor will record events when a unit of work completes. All records are collected in the unformatted event table.

Invocation

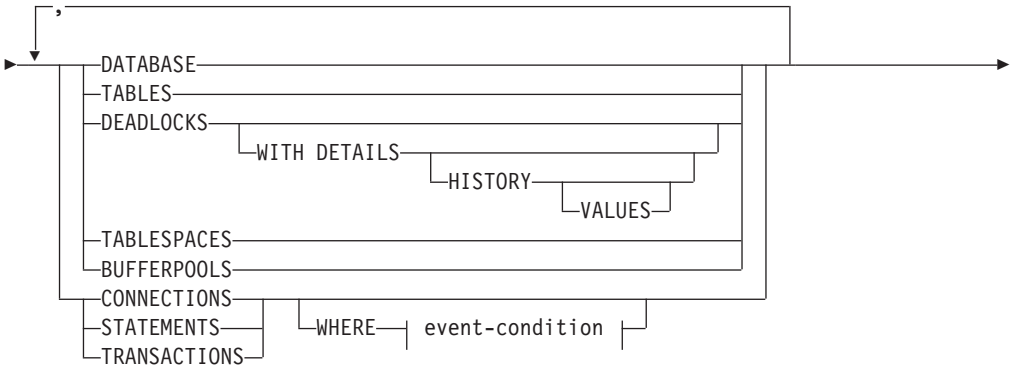
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

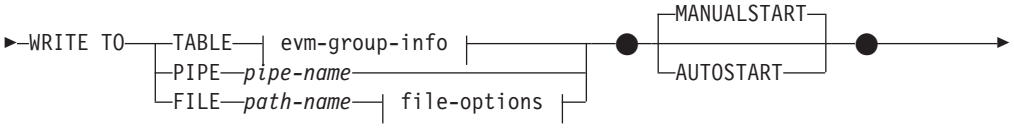
Authorization

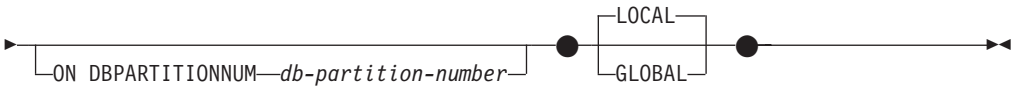
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority

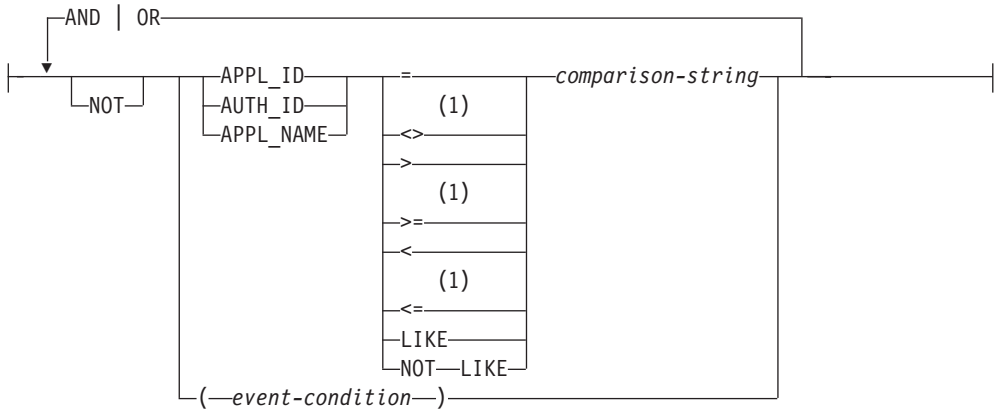
Syntax



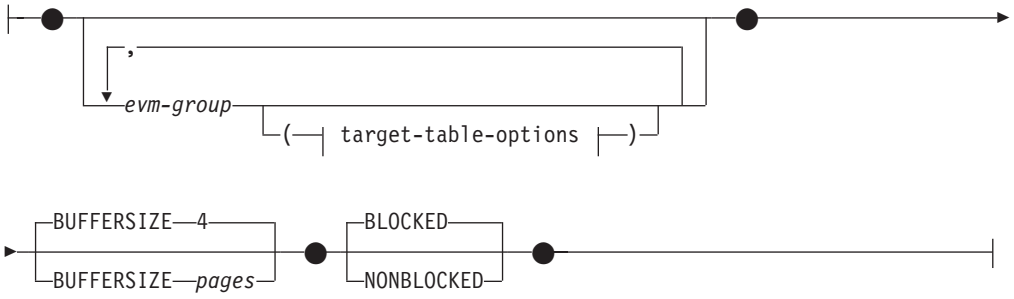




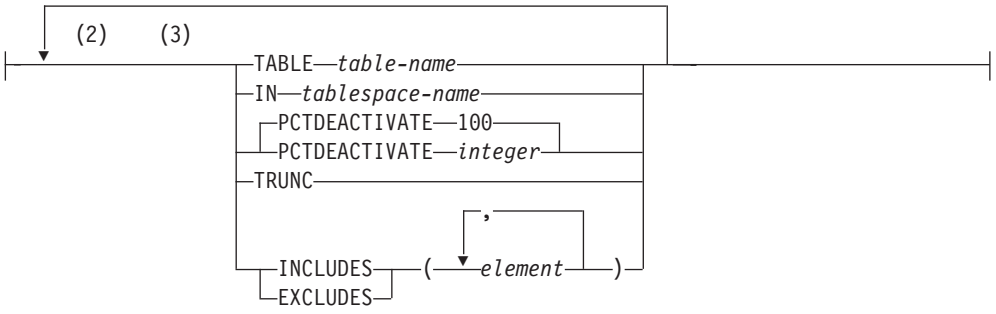
event-condition:



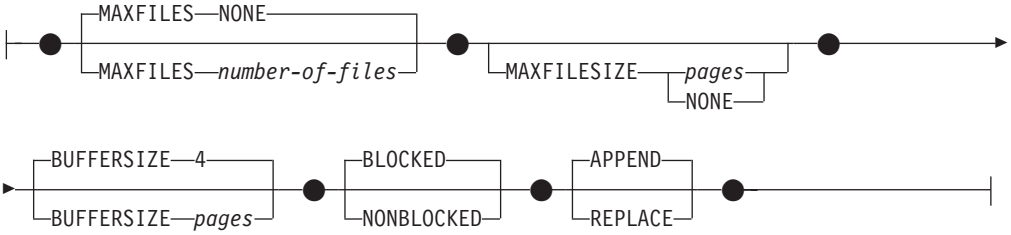
evm-group-info:



target-table-options:



file-options:



Notes:

1 Other forms of these operators are also supported.

2 Each clause can be specified only once.

3 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

DATABASE
Specifies that the event monitor records a database event when the last application disconnects from the database.

TABLES
Specifies that the event monitor records a table event for each active table when the last application disconnects from the database. For partitioned tables, a table event is recorded for each data partition of each active table. An active table is a table that has changed since the first connection to the database.

DEADLOCKS

Note: This option has been deprecated. Its use is no longer recommended and might be removed in a future release. Use the CREATE EVENT MONITOR FOR LOCKING statement to monitor lock-related events, such as lock timeouts, lock waits, and deadlocks.

Specifies that the event monitor records a deadlock event whenever a deadlock occurs.

WITH DETAILS
Specifies that the event monitor is to generate a more detailed deadlock connection event for each application that is involved in a deadlock. This additional detail includes:

- Information about the statement that the application was executing when the deadlock occurred, such as the statement text
- The locks held by the application when the deadlock occurred. In a partitioned database environment, this includes only those locks that are held on the database partition on which the application was waiting for its lock when the deadlock occurred. For partitioned tables, this includes the data partition identifier.

HISTORY
Specifies that the event monitor data will also include:

- The history of all statements in the current unit of work at the participating node (including WITH HOLD cursors opened in previous units of work). SELECT statements issued at the uncommitted read (UR) isolation level are not included in the statement history.
- The statement compilation environment for each SQL statement in binary format (if available)

VALUES
Specifies that the event monitor data will also include:

- The data values used as input variables for each SQL statement. These data values will not include LOB data, long data, structured type data, or XML data.

Only one of: DEADLOCKS, DEADLOCKS WITH DETAILS, DEADLOCKS WITH DETAILS HISTORY, or DEADLOCKS WITH DETAILS HISTORY VALUES can be specified in a single CREATE EVENT MONITOR statement (SQLSTATE 42613).

TABLESPACES
Specifies that the event monitor records a table space event for each table space when the last application disconnects from the database.

BUFFERPOOLS
Specifies that the event monitor records a buffer pool event when the last application disconnects from the database.

CONNECTIONS
Specifies that the event monitor records a connection event when an application disconnects from the database.

STATEMENTS
Specifies that the event monitor records a statement event whenever a SQL statement finishes executing.

TRANSACTIONS

Note: This option has been deprecated. Its use is no longer recommended and might be removed in a future release. Use the CREATE EVENT MONITOR FOR UNIT OF WORK statement to monitor transaction events.

Specifies that the event monitor records a transaction event whenever a transaction completes (that is, whenever there is a commit or rollback operation).

WHERE event-condition
Defines a filter that determines which connections cause a CONNECTION, STATEMENT or TRANSACTION event to occur. If the result of the event condition is TRUE for a particular connection, then that connection will generate the requested events.

This clause is a special form of the WHERE clause that should not be confused with a standard search condition.

To determine if an application will generate events for a particular event monitor, the WHERE clause is evaluated:

- For each active connection when an event monitor is first turned on
- Subsequently for each new connection to the database at connect time

The WHERE clause is not evaluated for each event.

If no WHERE clause is specified, all events of the specified event type will be monitored.

The event-condition must not exceed 32 678 bytes in length in the database code page (SQLSTATE 22001).

APPL_ID
Specifies that the application ID of each connection should be compared with the *comparison-string* in order to determine if the connection should generate CONNECTION, STATEMENT or TRANSACTION events (whichever was specified).

AUTH_ID
Specifies that the authorization ID of each connection should be compared with the *comparison-string* in order to determine if the connection should generate CONNECTION, STATEMENT or TRANSACTION events (whichever was specified).

APPL_NAME
Specifies that the application program name of each connection should be compared with the *comparison-string* in order to determine if the connection should generate CONNECTION, STATEMENT or TRANSACTION events (whichever was specified).

The application program name is the first 20 bytes of the application program file name, after the last path separator.

comparison-string
A string to be compared with the APPL_ID, AUTH_ID, or APPL_NAME of each application that connects to the database. *comparison-string* must be a string constant (that is, host variables and other string expressions are not permitted).

WRITE TO
Introduces the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of database tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is

mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

evm-group-info
Defines the target table for a logical data group. This clause should be specified for each grouping that is to be recorded. However, if no evm-group-info clauses are specified, all groups for the event monitor type are recorded.

evm-group
Identifies the logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Table 16. Values for evm-group based on the type of event monitor

Type of Event Monitor	evm-group Value
Database	• DB • CONTROL[1] • DBMEMUSE
Tables	• TABLE • CONTROL[1]
Deadlocks	• CONNHEADER • DEADLOCK • DLCONN • CONTROL[1]
Deadlocks with details	• CONNHEADER • DEADLOCK • DLCONN[2] • DLLOCK[3] • CONTROL[1]
Deadlocks with details history	• CONNHEADER • DEADLOCK • DLCONN[2] • DLLOCK[3] • STMTHIST • CONTROL[1]
Deadlocks with details history values	• CONNHEADER • DEADLOCK • DLCONN[2] • DLLOCK[3] • STMTHIST • STMTVALS • CONTROL[1]
Tablespaces	• TABLESPACE • CONTROL[1]
Bufferpools	• BUFFERPOOL • CONTROL[1]

Table 16. Values for evm-group based on the type of event monitor (continued)

Type of Event Monitor	evm-group Value
Connections	• CONNHEADER • CONN • CONTROL[1] • CONNMEMUSE
Statements	• CONNHEADER • STMT • SUBSECTION[4] • CONTROL[1]
Transactions	• CONNHEADER • XACT • CONTROL[1]
Activities	• ACTIVITY • ACTIVITYMETRICS • ACTIVITYSTMT • ACTIVITYVALS • CONTROL[1]
Statistics	• QSTATS • SCSTATS • SCMETRICS • WCSTATS • WLSTATS • WLMETRICS • HISTOGRAMBIN • CONTROL[1]
Threshold Violations	• THRESHOLDVIOLATIONS • CONTROL[1]
Locking[5]	• LOCK • LOCK_PARTICIPANTS • LOCK_PARTICIPANT_ACTIVITIES • LOCK_ACTIVITY_VALUES • CONTROL[1]
Package Cache[5]	• PKGCACHE • PKGCACHE_METRICS • CONTROL[1]
Unit of Work[5]	• UOW • UOW_METRICS • UOW_PACKGE_LIST • UOW_EXECUTABLE_LIST • CONTROL[1]

Table 16. Values for evm-group based on the type of event monitor (continued)

Type of Event Monitor	evm-group Value
Change History	• CHANGESUMMARY • EVMONSTART • TXNCOMPLETION • DDLSTMTEXEC • DBDBMCFG • REGVAR • UTILSTART • UTILSTOP • UTILPHASE • UTILLOCATION • CONTROL[1]

[1] Logical data groups dbheader (conn_time element only), start and overflow, are all written to the CONTROL group. The overflow group is written if the event monitor is non-blocked and events were discarded.

[2] Corresponds to the DETAILED_DLCONN event.

[3] Corresponds to the LOCK logical data groups that occur within each DETAILED_DLCONN event.

[4] Created only for partitioned database environments.

[5] Refers to the Formatted Event Table version of this event monitor type.

target-table-options

Identifies the target table for the group. If a value for *target-table-options* is not specified, CREATE EVENT MONITOR processing proceeds as follows:

- A derived table name is used (see description for TABLE *table-name*).
- A default table space is chosen (see description for IN *tablespace-name*).
- All elements are included.
- PCTDEACTIVATE and TRUNC are not specified.

TABLE *table-name*

Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If no name is provided, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
substring(evm-group CONCAT '_'
  CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*

Defines the table space in which the table is to be created. If no table space name is provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100, where 100 means that the event monitor deactivates when the table space becomes completely full. The default value assumed is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might deactivate unexpectedly if the table space reaches the threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

It is recommended that, when a target table space has auto-resize enabled, the PCTDEACTIVATE parameter be set to 100.

TRUNC
Specifies that the STMT_TEXT and STMT_VALUE_DATA columns are defined as VARCHAR(*n*), where *n* is the largest size that can fit into the table row. In this case, any data that is longer than *n* bytes is truncated. The following example illustrates how the value of *n* is calculated. Assume that:

- The table is created in a table space that uses 32K pages.
- The total length of all the other columns in the table equals 357 bytes.

In this case, the maximum row size for a table is 32677 bytes. Therefore, the element would be defined as VARCHAR(32316); that is, 32677 - 357 - 4. If TRUNC is not specified, the column will be defined as CLOB(2M). Note that STMT_TEXT is found in the STMT event group, the STMT_HISTORY event group, and the DLCONN event group (for deadlocks with details event monitors). STMT_VALUE_DATA is found in the DATA_VALUE event group.

INCLUDES
Specifies that the following elements are to be included in the table.

EXCLUDES
Specifies that the following elements are *not* to be included in the table.

element
Identifies a monitor element. Element information can be provided in one of the following forms:

- Specify no element information. In this case, all elements are included in the CREATE TABLE statement.

- Specify the elements to include in the form: INCLUDES (element1, element2, ..., element*n*). Only table columns are created for these elements.
- Specify the elements to exclude in the form: EXCLUDES (element1, element2, ..., element*n*). Only table columns are created for all elements except these.

Use the db2evtbl command to build a CREATE EVENT MONITOR statement that includes a complete list of elements for a group.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K pages). Table event monitors insert all data from a buffer, and issues a COMMIT once the buffer has been processed. The larger the buffers, the larger the commit scope used by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event monitors. When a monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O.

The default size of each buffer is 4 pages (two 16K buffers are allocated). The minimum size is 1 page. The maximum size of the buffers is limited by the size of the monitor heap, because the buffers are allocated from that heap. If many event monitors are being used at the same time, increase the size of the **mon_heap_sz** database manager configuration parameter.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

NONBLOCKED
Specifies that each agent that generates an event should not wait for the event buffer to be written out to disk if the agent determines that both event buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of BLOCKED event monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems.

PIPE
Specifies that the target for the event monitor data is a named pipe. The event monitor writes the data to the pipe in a single stream (that is, as if it were a single, infinitely long file). When writing the data to a pipe, an event monitor does not perform blocked writes. If there is no room in the pipe buffer, then the event monitor will discard the data. It is the monitoring application's responsibility to read the data promptly if it wishes to ensure no data loss.

pipe-name
The name of the pipe (FIFO on AIX) to which the event monitor will write the data.

The naming rules for pipes are platform specific. On UNIX operating systems, pipe names are treated like file names. As a result, relative pipe names are permitted, and are treated like relative path-names (see

description for *path-name*). On Windows, however, there is a special syntax for a pipe name and, as a result, absolute pipe names are required.

The existence of the pipe will not be checked at event monitor creation time. It is the responsibility of the monitoring application to have created and opened the pipe for reading at the time that the event monitor is activated. If the pipe is not available at this time, then the event monitor will turn itself off, and will log an error. (That is, if the event monitor was activated at database start time as a result of the AUTOSTART option, then the event monitor will log an error in the system error log.) If the event monitor is activated via the SET EVENT MONITOR STATE SQL statement, then that statement will fail (SQLSTATE 58030).

In a DB2 pureScale environment, the *pipe-name* must be on a shared file system whether this is a LOCAL or GLOBAL event monitor. This requirement is to allow these event monitors to operate correctly in the event of a member failover. Failure to use a *pipe-name* on a shared file system will result in an error (SQLSTATE 428A3) if the event monitor activates during a member failover.

FILE

Indicates that the target for the event monitor data is a file (or set of files). The event monitor writes out the stream of data as a series of 8 character numbered files, with the extension "evt". (for example, 00000000.evt, 00000001.evt, and 00000002.evt). The data should be considered to be one logical file even though the data is broken up into smaller pieces (that is, the start of the data stream is the first byte in the file 00000000.evt; the end of the data stream is the last byte in the file nnnnnnnn.evt).

The maximum size of each file can be defined as well as the maximum number of files. An event monitor will never split a single event record across two files. However, an event monitor may write related records in two different files. It is the responsibility of the application that uses this data to keep track of such related information when processing the event files.

path-name

The name of the directory in which the event monitor should write the event files data. The path must be known at the server; however, the path itself could reside on another database partition (for example, on a UNIX system, this might be an NFS mounted file). A string constant must be used when specifying the *path-name*.

The directory does not have to exist at CREATE EVENT MONITOR time. However, a check is made for the existence of the target path when the event monitor is activated. At that time, if the target path does not exist, an error (SQLSTATE 428A3) is raised.

If an absolute path (a path that starts with the root directory on AIX, or a disk identifier on Windows) is specified, the specified path will be the one used. In environments other than DB2 pureScale, if a relative path (a path that does not start with the root) is specified, then the path relative to the DB2EVENT directory in the database directory will be used. In a DB2 pureScale environment, if a relative path is specified, then the path relative to the database owning directory in the database directory will be used.

It is possible to specify two or more event monitors that have the same target path. However, once one of the event monitors has been activated for the first time, and as long as the target directory is not empty, it will be impossible to activate any of the other event monitors.

In a DB2 pureScale environment, the *path-name* must be on a shared file system whether this is a LOCAL or GLOBAL event monitor. This requirement is to allow these event monitors to operate correctly in the event of a member failover. Failure to use a *path-name* on a shared file system will result in an error (SQLSTATE 428A3) if the event monitor activates during a member failover.

file-options
Specifies the options for the file format.

MAXFILES NONE
Specifies that there is no limit to the number of event files that the event monitor will create. This is the default.

MAXFILES *number-of-files*
Specifies that there is a limit on the number of event monitor files that will exist for a particular event monitor at any time. Whenever an event monitor has to create another file, it will check to make sure that the number of .evt files in the directory is less than *number-of-files*. If this limit has already been reached, then the event monitor will turn itself off.

If an application removes the event files from the directory after they have been written, then the total number of files that an event monitor can produce can exceed *number-of-files*. This option has been provided to allow a user to guarantee that the event data will not consume more than a specified amount of disk space.

MAXFILESIZE *pages*
Specifies that there is a limit to the size of each event monitor file. Whenever an event monitor writes a new event record to a file, it checks that the file will not grow to be greater than *pages* (in units of 4K pages). If the resulting file would be too large, then the event monitor switches to the next file. The default for this option is:

- Windows - 200 4K pages
- UNIX - 1000 4K pages

The number of pages must be greater than at least the size of the event buffer in pages. If this requirement is not met, then an error (SQLSTATE 428A4) is raised.

MAXFILESIZE NONE
Specifies that there is no set limit on a file's size. If MAXFILESIZE NONE is specified, then MAXFILES 1 must also be specified. This option means that one file will contain all of the event data for a particular event monitor. In this case the only event file will be 00000000.evt.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K pages). All event monitor file I/O is buffered to improve the performance of the event monitors. The larger the buffers, the less I/O will be performed by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event

monitors. When the monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O.

The default size of each buffer is 4 pages (two 16K buffers are allocated). The minimum size is 1 page. The maximum size of the buffers is limited by the value of the MAXFILESIZE parameter, as well as the size of the monitor heap, because the buffers are allocated from that heap. If many event monitors are being used at the same time, increase the size of the **mon_heap_sz** database manager configuration parameter.

Event monitors that write their data to a pipe also have two internal (non-configurable) buffers that are each 1 page in size. These buffers are also allocated from the monitor heap (MON_HEAP). For each active event monitor that has a pipe target, increase the size of the database heap by 2 pages.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

NONBLOCKED
Specifies that each agent that generates an event should not wait for the event buffer to be written out to disk if the agent determines that both event buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of BLOCKED event monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems.

APPEND
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will append the new event data to the existing stream of data files. When the event monitor is reactivated, it will resume writing to the event files as if it had never been turned off. APPEND is the default option.

The APPEND option does not apply at CREATE EVENT MONITOR time, if there is existing event data in the directory where the newly created event monitor is to write its event data.

REPLACE
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will erase all of the event files and start writing data to file 00000000.evt.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor has been activated, it can be deactivated only by using the SET EVENT MONITOR STATE statement or by stopping the instance. This is the default.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the database partition on which the event monitor runs is activated.

ON DBPARTITIONNUM *db-partition-number*
Specifies the database partition (in a partitioned database environment) or member (in a DB2 pureScale environment) on which a file or pipe event

monitor is to run. When the monitoring scope is defined as LOCAL, data is collected only on the specified partition or member. When the monitoring scope is defined as GLOBAL, all database partitions or members collect data and report to the database partition or member with the specified number. The I/O component will physically run on the specified database partition or member, writing records to the specified file or pipe. When DB2 pureScale is enabled, -1 can be specified, which allows the I/O component to run from any active member. Additionally, in the event that the I/O component is no longer able to run on a given member, the event monitor will be restarted with the I/O component running on another available active member.

This clause is not valid for table event monitors. In a partitioned database environment, write-to-table event monitors will run and write events on all database partitions where table spaces for target tables are defined. In a DB2 pureScale environment, write-to-table event monitors will record events on all active members.

If this clause is not specified and DB2 pureScale is not enabled, the currently connected database partition number (for the application) is used. If this clause is not specified and DB2 pureScale is enabled, the I/O component is able to run on any currently connected database partition number.

LOCAL
The event monitor reports only on the database partition that is running. It gives a partial trace of the database activity. This is the default.

This clause is valid for file or pipe monitors. It is not valid for table event monitors.

GLOBAL
The event monitor reports on all database partitions. For a partitioned database, only DEADLOCKS event monitors can be defined as GLOBAL.

This clause is valid for file or pipe monitors. It is not valid for table event monitors.

Rules

- Each of the event types (DATABASE, TABLES, DEADLOCKs,...) can only be specified once in a particular event monitor definition.

Notes

- Event monitor definitions are recorded in the SYSCAT.EVENTMONITORS catalog view. The events themselves are recorded in the SYSCAT.EVENTS catalog view. The names of target tables are recorded in the SYSCAT.EVENTTABLES catalog view.
- There is a performance impact when using DEADLOCKS WITH DETAILS rather than DEADLOCKS. When a deadlock occurs, the database manager requires extra time to record the extra deadlock information.
- A CONNHEADER event is normally written whenever a connection is established. However, if an event monitor is created only for DEADLOCKS WITH DETAILS, a CONNHEADER event will only be written the first time that the connection participates in a deadlock.
- In a database with multiple database partitions, the ON DBPARTITIONNUM clause can be used with FILE and PIPE event monitors having a DEADLOCKS event type to indicate where the event monitor itself should reside; information from other database partitions, if relevant, is sent to that location for processing.

- In a database with multiple database partitions, a deadlock event monitor will receive information about applications that have locks participating in the deadlock from all the database partitions on which those participating locks existed. If the database partition to which the application is connected (the application coordinator partition) is not one of the participating database partitions, no information about a deadlock event will be received from that database partition.
- The BUFFERSIZE parameter restricts the size of STMT, STMT_HISTORY, DATA_VALUE, and DETAILED_DLCONN events. If a STMT or a STMT_HISTORY event cannot fit within a buffer, it is truncated by truncating statement text. If a DETAILED_DLCONN event cannot fit within a buffer, it is truncated by removing locks. If it still cannot fit, statement text is truncated. If a DATA_VAL event cannot fit within a buffer, the data value is truncated.

  Event monitors WITH DETAILS HISTORY VALUES (and, to a lesser extent, WITH DETAILS HISTORY) use a significant amount of monitor heap space to track statements and their data values. For more information, see the description of the **mon_heap_sz** database manager configuration parameter.
- If the database partition on which the event monitor is to run is not active, event monitor activation occurs when that database partition next activates.
- After an event monitor is activated, it behaves like an autostart event monitor until that event monitor is explicitly deactivated or the instance is recycled. That is, if an event monitor is active when a database partition is deactivated, and that database partition is subsequently reactivated, the event monitor is also explicitly reactivated.
- ***Write to table event monitors***: General notes:
  - All target tables are created when the CREATE EVENT MONITOR statement executes.
  - If the creation of a table fails for any reason, an error is passed back to the application program, and the CREATE EVENT MONITOR statement fails.
  - A target table can only be used by one event monitor. During CREATE EVENT MONITOR processing, if a target table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view.
  - During CREATE EVENT MONITOR processing, if a table already exists, but is *not* defined for use by another event monitor, no table is created, and processing continues. A warning is passed back to the application program.
  - Any table spaces must exist before the CREATE EVENT MONITOR statement is executed. The CREATE EVENT MONITOR statement does not create table spaces.
  - If specified, the LOCAL and GLOBAL keywords are ignored. With WRITE TO TABLE event monitors, an event monitor output process or thread is started on each database partition in the instance, and each of these processes reports data only for the database partition on which it is running.
  - The following event types from the flat monitor log file or pipe format are not recorded by write to table event monitors:
    - LOG_STREAM_HEADER
    - LOG_HEADER
    - DB_HEADER (Elements db_name and db_path are not recorded. The element conn_time is recorded in CONTROL.)

– In a partitioned database environment, data is only written to target tables on the database partitions where their table spaces exist. If a table space for a target table does not exist on some database partition, data for that target table is ignored. This behavior allows users to choose a subset of database partitions for monitoring, by creating a table space that exists only on certain database partitions. In a DB2 pureScale environment, data will be written from every member.

  In a partitioned database environment, if some target tables do not reside on a database partition, but other target tables do reside on that same database partition, only the data for the target tables that do reside on that database partition is recorded.
– Users must manually prune all target tables.

Table Columns:

– Column names in a table match an event monitor element identifier. Any event monitor element that does not have a corresponding target table column is ignored.
– Use the db2evtbl command to build a CREATE EVENT MONITOR command that includes a complete list of elements for a group.
– The types of columns being used for monitor elements correlate to the following mapping:

```
SQLM_TYPE_STRING                    CHAR[n], VARCHAR[n] or CLOB(n)
                                    (If the data in the event monitor
                                    record exceeds n bytes,
                                    it is truncated.)
SQLM_TYPE_U8BIT and SQLM_TYPE_8BIT  SMALLINT, INTEGER or BIGINT
SQLM_TYPE_16BIT and SQLM_TYPE_U16BIT SMALLINT, INTEGER or BIGINT
SQLM_TYPE_32BIT and SQLM_TYPE_U32BIT INTEGER or BIGINT
SQLM_TYPE_U64BIT and SQLM_TYPE_64BIT BIGINT
sqlm_timestamp                      TIMESTAMP
sqlm_time(elapsed time)             BIGINT
sqlca:
  sqlerrmc                          VARCHAR[72]
  sqlstate                          CHAR[5]
  sqlwarn                           CHAR[11]
  other fields                      INTEGER or BIGINT
```

– Columns are defined to be NOT NULL.
– Because the performance of tables with CLOB columns is inferior to tables that have VARCHAR columns, consider using the TRUNC keyword when specifying the STMT *evm-group* value (or the DLCONN *evm-group* value, if using the DEADLOCKS WITH DETAILS event type).
– Unlike other target tables, the columns in the CONTROL table do not match monitor element identifiers. Columns are defined as follows:

```
Column Name      Data Type     Nullable  Description
-----------      ---------     --------  -----------
PARTITION_KEY    INTEGER       N         Distribution key (partitioned
                                         database only)
PARTITION_NUMBER INTEGER       N         Database partition number
                                         (partitioned database only)
EVMONNAME        VARCHAR(128)  N         Name of the event monitor
MESSAGE          VARCHAR(128)  N         Describes the nature of
                                         the MESSAGE_TIME column.
                                         This can be one of
                                         the following values:
                                         - FIRST_CONNECT (the time
                                           of the first connect to the
                                           database after activation)
                                         - EVMON_START (the time that
                                           the event monitor listed
```

```
                                          in EVMONNAME was started)
                                        - OVERFLOWS:n (denotes that
                                          n records were discarded
                                          because of buffer overflow)
                                        - LAST_DROPPED_RECORD
                                          (the last time that
                                          an overflow occurred)
MESSAGE_TIME   TIMESTAMP       N        Timestamp
```

– In a partitioned database environment, the first column of each table is named PARTITION_KEY, is NOT NULL, and is of type INTEGER. This column is used as the distribution key for the table. The value of this column is chosen so that each event monitor process inserts data into the database partition on which the process is running; that is, insert operations are performed locally on the database partition where the event monitor process is running. On any database partition, the PARTITION_KEY field will contain the same value. This means that if a database partition is dropped and data redistribution is performed, all data on the dropped database partition will go to one other database partition instead of being evenly distributed. Therefore, before removing a database partition, consider deleting all table rows on that database partition.
– In a partitioned database environment, a column named PARTITION_NUMBER can be defined for each table. This column is NOT NULL and is of type INTEGER. It contains the number of the database partition on which the data was inserted. Unlike the PARTITION_KEY column, the PARTITION_NUMBER column is not mandatory. The PARTITION_NUMBER column is not allowed in a non-partitioned database environment.

Table Attributes:

– Default table attributes are used. Besides distribution key (partitioned databases only), no extra options are specified when creating tables.
– Indexes on the table can be created.
– Extra table attributes (such as volatile, RI, triggers, constraints, and so on) can be added, but the event monitor process (or thread) will ignore them.
– If "not logged initially" is added as a table attribute, it is turned off at the first COMMIT, and is not set back on.

Event Monitor Activation:

– When an event monitor activates, all target table names are retrieved from the SYSCAT.EVENTTABLES catalog view.
– In a partitioned database environment, activation processing occurs on every database partition of the instance. On a particular database partition, activation processing determines the table spaces and database partition groups for each target table. The event monitor only activates on a database partition if at least one target table exists on that database partition. Moreover, if some target table is not found on a database partition, that target table is flagged so that data destined for that table is dropped during runtime processing.
– If a target table does not exist when the event monitor activates (or, in a partitioned database environment, if the table space does not reside on a database partition), activation continues, and data that would otherwise be inserted into this table is ignored.
– Activation processing validates each target table. If validation fails, activation of the event monitor fails, and messages are written to the administration log.

  - During activation in a partitioned database environment, the CONTROL table rows for FIRST_CONNECT and EVMON_START are only inserted on the catalog database partition. This requires that the table space for the control table exist on the catalog database partition. If it does not exist on the catalog database partition, these inserts are not performed.
  - In a partitioned database environment, if a partition is not yet active when a write to table event monitor is activated, the event monitor will be activated the next time that partition is activated.

  Run Time:

  - An event monitor runs with DATAACCESS authority.
  - If, while an event monitor is active, an insert operation into a target table fails:
    - Uncommitted changes are rolled back.
    - A message is written to the administration log.
    - The event monitor is deactivated.
  - If an event monitor is active, it performs a local COMMIT when it has finished processing an event monitor buffer.
  - In a partitioned database environment, the actual statement text, which can be up to 2MB in length, is only stored (in the STMT or DLCONN table) by the event monitor process running on the application coordinator database partition. On other database partitions, this value has zero length.
  - In an environment other than a partitioned database or a DB2 pureScale database, all write to table event monitors are deactivated when the last application terminates (and the database has not been explicitly activated). In a DB2 pureScale environment, write to table event monitors are deactivated on a given member when the database deactivates on that member and reactivates when the database activates on that member again. In a partitioned database environment, write to table event monitors are deactivated when the catalog partition deactivates.
  - The DROP EVENT MONITOR statement does not drop target tables.
  - Whenever a write-to-table event monitor activates, it will acquire IN table locks on each target table in order to prevent them from being modified while the event monitor is active. Table locks are maintained on all tables while the event monitor is active. If exclusive access is required on any of the target tables (for example, when a utility is to be run), first deactivate the event monitor to release the table locks before attempting such access.

- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NODE can be specified in place of DBPARTITIONNUM
  - Commas can be used to separate multiple options in the *target-table-options* clause.

Examples

- *Example 1:* The following example creates an event monitor called SMITHPAY. This event monitor, will collect event data for the database as well as for the SQL statements performed by the PAYROLL application owned by the JSMITH authorization ID. The data will be appended to the absolute path /home/jsmith/event/smithpay/. A maximum of 25 files will be created. Each file will be a maximum of 1 024 4K pages long. The file I/O will be non-blocked.

```
CREATE EVENT MONITOR SMITHPAY
  FOR DATABASE, STATEMENTS
  WHERE APPL_NAME = 'PAYROLL' AND AUTH_ID = 'JSMITH'
  WRITE TO FILE '/home/jsmith/event/smithpay'
  MAXFILES 25
  MAXFILESIZE 1024
  NONBLOCKED
  APPEND
```

- *Example 2:* The following example creates an event monitor called DEADLOCKS_EVTS. This event monitor will collect deadlock events and will write them to the relative path DLOCKS. One file will be written, and there is no maximum file size. Each time the event monitor is activated, it will append the event data to the file 00000000.evt if it exists. The event monitor will be started each time the database is started. The I/0 will be blocked by default.

  ```
  CREATE EVENT MONITOR DEADLOCK_EVTS
    FOR DEADLOCKS
    WRITE TO FILE 'DLOCKS'
    MAXFILES 1
    MAXFILESIZE NONE
    AUTOSTART
  ```

- *Example 3:* This example creates an event monitor called DB_APPLS. This event monitor collects connection events, and writes the data to the named pipe /home/jsmith/applpipe.

  ```
  CREATE EVENT MONITOR DB_APPLS
    FOR CONNECTIONS
    WRITE TO PIPE '/home/jsmith/applpipe'
  ```

- *Example 4:* This example, which assumes a partitioned database environment, creates an event monitor called FOO. This event monitor collects SQL statement events and writes them to SQL tables with the following derived names:
  - CONNHEADER_FOO
  - STMT_FOO
  - SUBSECTION_FOO
  - CONTROL_FOO

  Because no table space information is supplied, all tables will be created in a table space selected by the system, based on the rules described under the IN *tablespace-name* clause. All tables include all elements for their group (that is, columns are defined whose names are equivalent to the element names.)

  ```
  CREATE EVENT MONITOR FOO
    FOR STATEMENTS
    WRITE TO TABLE
  ```

- *Example 5:* This example, which assumes a partitioned database environment, creates an event monitor called BAR. This event monitor collects SQL statement and transaction events and writes them to tables as follows:
  - Any data from the STMT group is written to table MYDEPT.MYSTMTINFO. The table is created in table space MYTABLESPACE. Create columns only for the following elements: ROWS_READ, ROWS_WRITTEN, and STMT_TEXT. Any other elements of the group will be discarded.
  - Any data from the SUBSECTION group is written to table MYDEPT.MYSUBSECTIONINFO. The table is created in table space MYTABLESPACE. The table includes all columns, except START_TIME, STOP_TIME, and PARTIAL_RECORD.
  - Any data from the XACT group is written to table XACT_BAR. Because no table space information is supplied, the table will be created in a table space

selected by the system, based on the rules described under the IN *tablespace-name* clause. This table includes all elements contained in the XACT group.

– No tables are created for connheader or control; all data for these groups are discarded.

```
CREATE EVENT MONITOR BAR
  FOR STATEMENTS, TRANSACTIONS
  WRITE TO TABLE
  STMT(TABLE MYDEPT.MYSTMTINFO IN MYTABLESPACE
    INCLUDES(ROWS_READ, ROWS_WRITTEN, STMT_TEXT)),
  STMT(TABLE MYDEPT.MYSTMTINFO IN MYTABLESPACE
    EXCLUDES(START_TIME, STOP_TIME, PARTIAL_RECORD)),
  XACT
```

CREATE EVENT MONITOR (activities)

The CREATE EVENT MONITOR (activities) statement defines a monitor that will record activity events that occur when using the database. The definition of the activity event monitor also specifies where the database should record the events.

Invocation

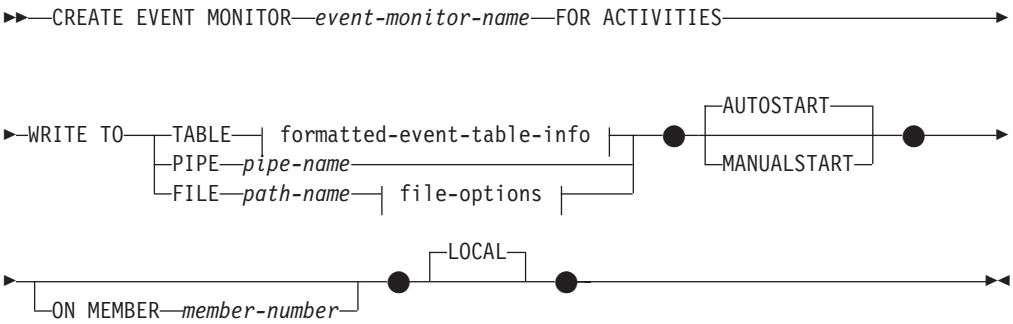
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

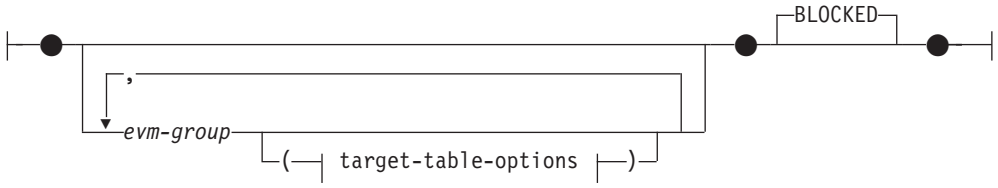
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority
- WLMADM authority

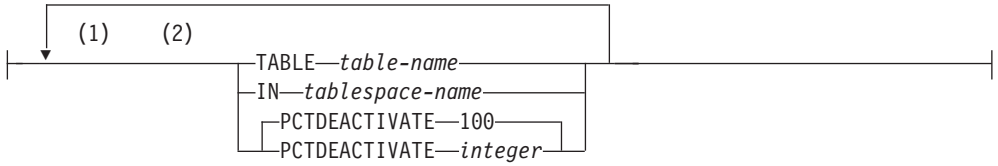
Syntax



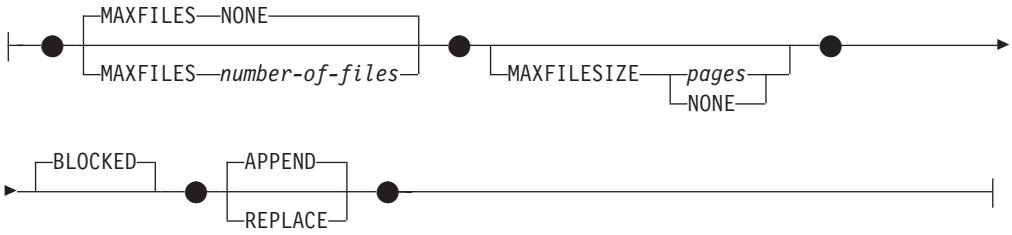
formatted-event-table-info:



target-table-options:



file-options:



Notes:

1 Each clause can be specified only once.

2 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

ACTIVITIES
Specifies that the event monitor records an activity event when an activity finishes executing, or before the completion of execution if the event is triggered by the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure. The activity must either:

- Belong to a service class or workload that has COLLECT ACTIVITY DATA set
- Belong to a work class whose associated work action is COLLECT ACTIVITY DATA
- Be identified as the activity that violated a threshold whose COLLECT ACTIVITY DATA clause was specified
- Have been identified in a call to the WLM_CAPTURE_ACTIVITY_IN_PROGRESS procedure before completing

WRITE TO
Introduces the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of database tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

formatted-event-table-info
Defines the target tables for an event monitor. This clause should be

specified for each grouping that is to be recorded. However, if no evm-group-info clauses are specified, all groups for the event monitor type are recorded.

evm-group
Identifies the logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Type of Event Monitor	evm-group Value
Activities	• ACTIVITY • ACTIVITYMETRICS • ACTIVITYSTMT • ACTIVITYVALS • CONTROL

target-table-options
Identifies the target table for the group.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If no name is provided, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
substring(evm-group CONCAT '_'
  CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*
Defines the table space in which the table is to be created. If no table space name is provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

Since the page size affects the INLINE LOB lengths used, consider specifying a table space with as large a page size as possible in order to improve the INSERT performance of the event monitor.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100, where 100 means that the event monitor deactivates when the table space becomes completely full. The default value assumed is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE parameter to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might deactivate unexpectedly if the table space reaches the

threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

If a value for *target-table-options* is not specified, CREATE EVENT MONITOR processing proceeds as follows:

- A derived table name is used.
- A default table space is chosen.
- The PCTDEACTIVATE parameter defaults to 100.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

PIPE
Specifies that the target for the event monitor data is a named pipe. The event monitor writes the data to the pipe in a single stream (that is, as if it were a single, infinitely long file). When writing the data to a pipe, an event monitor does not perform blocked writes. If there is no room in the pipe buffer, then the event monitor will discard the data. It is the monitoring application's responsibility to read the data promptly if it wishes to ensure no data loss.

pipe-name
The name of the pipe (FIFO on AIX) to which the event monitor will write the data.

The naming rules for pipes are platform specific. On UNIX operating systems, pipe names are treated like file names. As a result, relative pipe names are permitted, and are treated like relative path-names (refer to the description for *path-name*). On Windows, however, there is a special syntax for a pipe name and, as a result, absolute pipe names are required.

The existence of the pipe will not be checked at event monitor creation time. It is the responsibility of the monitoring application to have created and opened the pipe for reading at the time that the event monitor is activated. If the pipe is not available at this time, then the event monitor will turn itself off, and will log an error. (That is, if the event monitor was activated at database start time as a result of the AUTOSTART option, then the event monitor will log an error in the system error log.) If the event monitor is activated via the SET EVENT MONITOR STATE SQL statement, then that statement will fail (SQLSTATE 58030).

FILE
Indicates that the target for the event monitor data is a file (or set of files). The event monitor writes out the stream of data as a series of 8 character numbered files, with the extension "evt". (for example, `00000000.evt`, `00000001.evt`, and `00000002.evt`). The data should be considered to be one logical file even though the data is broken up into smaller pieces (that is, the start of the data stream is the first byte in the file `00000000.evt`; the end of the data stream is the last byte in the file *nnnnnnnn*`.evt`).

The maximum size of each file can be defined as well as the maximum number of files. An event monitor will never split a single event record across two files. However, an event monitor may write related records in

two different files. It is the responsibility of the application that uses this data to keep track of such related information when processing the event files.

path-name
The name of the directory in which the event monitor should write the event files data. The path must be known at the server; however, the path itself could reside on another database partition (for example, on a UNIX system, this might be an NFS mounted file). A string constant must be used when specifying the *path-name*.

The directory does not have to exist at CREATE EVENT MONITOR time. However, a check is made for the existence of the target path when the event monitor is activated. At that time, if the target path does not exist, an error (SQLSTATE 428A3) is raised.

If an absolute path (a path that starts with the root directory on AIX, or a disk identifier on Windows) is specified, the specified path will be the one used. In environments other than DB2 pureScale, if a relative path (a path that does not start with the root) is specified, then the path relative to the `DB2EVENT` directory in the database directory will be used. In a DB2 pureScale environment, if a relative path is specified, then the path relative to the database owning directory in the database directory will be used.

It is possible to specify two or more event monitors that have the same target path. However, once one of the event monitors has been activated for the first time, and as long as the target directory is not empty, it will be impossible to activate any of the other event monitors.

file-options
Specifies the options for the file format.

MAXFILES NONE
Specifies that there is no limit to the number of event files that the event monitor will create. This is the default.

MAXFILES *number-of-files*
Specifies that there is a limit on the number of event monitor files that will exist for a particular event monitor at any time. Whenever an event monitor has to create another file, it will check to make sure that the number of .evt files in the directory is less than *number-of-files*. If this limit has already been reached, then the event monitor will turn itself off.

If an application removes the event files from the directory after they have been written, then the total number of files that an event monitor can produce can exceed *number-of-files*. This option has been provided to allow a user to guarantee that the event data will not consume more than a specified amount of disk space.

MAXFILESIZE *pages*
Specifies that there is a limit to the size of each event monitor file. Whenever an event monitor writes a new event record to a file, it checks that the file will not grow to be greater than *pages* (in units of 4K pages). If the resulting file would be too large, then the event monitor switches to the next file. The default for this option is:

- Windows - 200 4K pages
- UNIX - 1000 4K pages

The number of pages must be greater than at least the size of the event buffer in pages. If this requirement is not met, then an error (SQLSTATE 428A4) is raised.

MAXFILESIZE NONE
Specifies that there is no set limit on a file's size. If MAXFILESIZE NONE is specified, then MAXFILES 1 must also be specified. This option means that one file will contain all of the event data for a particular event monitor. In this case the only event file will be 00000000.evt.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

APPEND
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will append the new event data to the existing stream of data files. When the event monitor is reactivated, it will resume writing to the event files as if it had never been turned off. APPEND is the default option.

The APPEND option does not apply at CREATE EVENT MONITOR time, if there is existing event data in the directory where the newly created event monitor is to write its event data.

REPLACE
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will erase all of the event files and start writing data to file `00000000.evt`.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor has been activated, it can be deactivated only by using the SET EVENT MONITOR STATE statement or by stopping the instance.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the database partition on which the event monitor runs is activated. This is the default behavior of the activities event monitor.

ON MEMBER *member-number*
Specifies the member on which a file or pipe event monitor is to run. When the monitoring scope is defined as LOCAL, data is collected only on the specified member. The I/O component will physically run on the specified member, writing records to the specified file or pipe. When the DB2 pureScale feature is enabled, -1 is the default. If a value of -1 is specified, it allows the I/O component to run from any active member. Additionally, in the event that the I/O component is no longer able to run on a given member, the event monitor will be restarted with the I/O component running on another available active member.

This clause is not valid for table event monitors. In a partitioned database environment, write-to-table event monitors will run and write events on all database partitions where table spaces for target tables are defined. In a DB2 pureScale environment, write-to-table event monitors will record events on all active members.

If this clause is not specified and DB2 pureScale is not enabled, the currently connected member (for the application) is used. If this clause is not specified and DB2 pureScale is enabled, the I/O component is able to run on any currently connected member.

LOCAL
The event monitor reports only on the member that is running. It gives a partial trace of the database activity. This is the default.

This clause is valid for file or pipe monitors. It is not valid for table event monitors.

GLOBAL is not a valid scope for this type of event monitor.

Rules

- The ACTIVITIES event type cannot be combined with any other event types in a particular event monitor definition.

Notes

- Event monitor definitions are recorded in the SYSCAT.EVENTMONITORS catalog view. The events themselves are recorded in the SYSCAT.EVENTS catalog view. The names of target tables are recorded in the SYSCAT.EVENTTABLES catalog view.
- If the member on which the event monitor is to run is not active, event monitor activation occurs when that member is reactivated.
- After an event monitor is activated, it behaves like an autostart event monitor until that event monitor is explicitly deactivated or the instance is recycled. That is, if an event monitor is active when a member is deactivated, and that member is subsequently reactivated, the event monitor is also explicitly reactivated.
- The FLUSH EVENT MONITOR statement is not applicable to this event monitor and will have no effect when issued against it.
- ***Write to table event monitors***: General notes:
  - All target tables are created when the CREATE EVENT MONITOR statement executes.
  - If the creation of a table fails for any reason, an error is passed back to the application program, and the CREATE EVENT MONITOR statement fails.
  - A target table can only be used by one event monitor. During CREATE EVENT MONITOR processing, if a target table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view.
  - During CREATE EVENT MONITOR processing, if a table already exists, but is *not* defined for use by another event monitor, no table is created, and processing continues. A warning is passed back to the application program.
  - Any table spaces must exist before the CREATE EVENT MONITOR statement is executed. The CREATE EVENT MONITOR statement does not create table spaces.
  - If specified, the LOCAL and GLOBAL keywords are ignored. With WRITE TO TABLE event monitors, an event monitor output process or thread is started on each member in the instance, and each of these processes reports data only for the member on which it is running.
  - The following event types from the flat monitor log file or pipe format are not recorded by write to table event monitors:

- LOG_STREAM_HEADER
- LOG_HEADER
- DB_HEADER (Elements **db_name** and **db_path** are not recorded. The element **conn_time** is recorded in CONTROL.)

– In a partitioned database environment, data is only written to target tables on the database partitions where their table spaces exist. If a table space for a target table does not exist on some database partition, data for that target table is ignored. This behavior allows users to choose a subset of database partitions for monitoring, by creating a table space that exists only on certain database partitions. In a DB2 pureScale environment, data will be written from every member.

  In a partitioned database environment, if some target tables do not reside on a database partition, but other target tables do reside on that same database partition, only the data for the target tables that do reside on that database partition is recorded.

– Users must manually prune all target tables.

Table Columns:

– Column names in a table match an event monitor element identifier. Any event monitor element that does not have a corresponding target table column is ignored.

– Use the **db2evtbl** command to build a CREATE EVENT MONITOR statement that includes a complete list of elements for a group.

– The types of columns being used for monitor elements correlate to the following mapping:

```
SQLM_TYPE_STRING                    CHAR[n], VARCHAR[n] or CLOB(n)
                                    (If the data in the event monitor
                                    record exceeds n bytes,
                                    it is truncated.)
SQLM_TYPE_U8BIT and SQLM_TYPE_8BIT  SMALLINT, INTEGER or BIGINT
SQLM_TYPE_16BIT and SQLM_TYPE_U16BIT SMALLINT, INTEGER or BIGINT
SQLM_TYPE_32BIT and SQLM_TYPE_U32BIT INTEGER or BIGINT
SQLM_TYPE_U64BIT and SQLM_TYPE_64BIT BIGINT
sqlm_timestamp                      TIMESTAMP
sqlm_time(elapsed time)             BIGINT
sqlca:
  sqlerrmc                          VARCHAR[72]
  sqlstate                          CHAR[5]
  sqlwarn                           CHAR[11]
  other fields                      INTEGER or BIGINT
```

– Columns are defined to be NOT NULL.

– Unlike other target tables, the columns in the CONTROL table do not match monitor element identifiers. Columns are defined as follows:

```
Column Name      Data Type     Nullable  Description
-----------      ---------     --------  -----------
PARTITION_KEY    INTEGER       N         Distribution key (partitioned
                                         database only)
PARTITION_NUMBER INTEGER       N         Database partition number
                                         (partitioned database only)
EVMONNAME        VARCHAR(128)  N         Name of the event monitor
MESSAGE          VARCHAR(128)  N         Describes the nature of
                                         the MESSAGE_TIME column.
                                         This can be one of
                                         the following values:
                                         - FIRST_CONNECT (the time
                                           of the first connect to the
                                           database after activation)
                                         - EVMON_START (the time that
```

```
                                          the event monitor listed
                                          in EVMONNAME was started)
                                        - OVERFLOWS:n (denotes that
                                          n records were discarded
                                          because of buffer overflow)
                                        - LAST_DROPPED_RECORD
                                          (the last time that
                                          an overflow occurred)
MESSAGE_TIME   TIMESTAMP       N        Timestamp
```

– In a partitioned database environment, the first column of each table is named PARTITION_KEY, is NOT NULL, and is of type INTEGER. This column is used as the distribution key for the table. The value of this column is chosen so that each event monitor process inserts data into the member on which the process is running; that is, insert operations are performed locally on the member where the event monitor process is running. On any database partition, the PARTITION_KEY field will contain the same value. This means that if a database partition is dropped and data redistribution is performed, all data on the dropped database partition will go to one other database partition instead of being evenly distributed. Therefore, before removing a database partition, consider deleting all table rows on that database partition.
– In a partitioned database environment, a column named PARTITION_NUMBER can be defined for each table. This column is NOT NULL and is of type INTEGER. It contains the number of the database partition on which the data was inserted. Unlike the PARTITION_KEY column, the PARTITION_NUMBER column is not mandatory. The PARTITION_NUMBER column is not allowed in a non-partitioned database environment.

Table Attributes:

– Default table attributes are used. Besides distribution key (partitioned databases only), no extra options are specified when creating tables.
– Indexes on the table can be created.
– Extra table attributes (such as volatile, RI, triggers, constraints, and so on) can be added, but the event monitor process (or thread) will ignore them.
– If "not logged initially" is added as a table attribute, it is turned off at the first COMMIT, and is not set back on.

Event Monitor Activation:

– When an event monitor activates, all target table names are retrieved from the SYSCAT.EVENTTABLES catalog view.
– In a partitioned database environment, activation processing occurs on every member of the instance. On a particular member, activation processing determines the table spaces and database partition groups for each target table. The event monitor only activates on a database partition if at least one target table exists on that database partition. Moreover, if some target table is not found on a database partition, that target table is flagged so that data destined for that table is dropped during runtime processing.
– If a target table does not exist when the event monitor activates (or, in a partitioned database environment, if the table space does not reside on a database partition), activation continues, and data that would otherwise be inserted into this table is ignored.
– Activation processing validates each target table. If validation fails, activation of the event monitor fails, and messages are written to the administration log.
– During activation in a partitioned database environment, the CONTROL table rows for FIRST_CONNECT and EVMON_START are only inserted on the catalog database partition. This requires that the table space for the control

table exist on the catalog database partition. If it does not exist on the catalog database partition, these inserts are not performed.
  - In a partitioned database environment, if a member is not yet active when a write to table event monitor is activated, the event monitor will be activated the next time that member is activated.

  Run Time:
  - An event monitor runs with DATAACCESS authority.
  - If, while an event monitor is active, an insert operation into a target table fails:
    - Uncommitted changes are rolled back.
    - A message is written to the administration log.
    - The event monitor is deactivated.
  - If an event monitor is active, it performs a local COMMIT when it has finished processing an event monitor buffer.
  - In an environment other than a partitioned database or a DB2 pureScale environment, all write to table event monitors are deactivated when the last application terminates (and the database has not been explicitly activated). In a DB2 pureScale environment, write to table event monitors are deactivated on a given member when the member stops and is reactivated when the member restarts. In a partitioned database environment, write to table event monitors are deactivated when the catalog partition deactivates.
  - The DROP EVENT MONITOR statement does not drop target tables.
  - Whenever a write-to-table event monitor activates, it will acquire IN table locks on each target table in order to prevent them from being modified while the event monitor is active. Table locks are maintained on all tables while the event monitor is active. If exclusive access is required on any of the target tables (for example, when a utility is to be run), first deactivate the event monitor to release the table locks before attempting such access.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DBPARTITIONNUM or NODE can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - Commas can be used to separate multiple options in the *target-table-options* clause

Example

Define an activity event monitor named DB2ACTIVITIES

```
CREATE EVENT MONITOR DB2ACTIVITIES
    FOR ACTIVITIES
    WRITE TO TABLE
    ACTIVITY (TABLE ACTIVITY_DB2ACTIVITIES
              IN USERSPACE1
              PCTDEACTIVATE 100),
    ACTIVITYMETRICS (TABLE ACTIVITYMETRICS_DB2ACTIVITIES
                     IN USERSPACE1
                     PCTDEACTIVATE 100),
    ACTIVITYSTMT (TABLE ACTIVITYSTMT_DB2ACTIVITIES
                  IN USERSPACE1
                  PCTDEACTIVATE 100),
    ACTIVITYVALS (TABLE ACTIVITYVALS_DB2ACTIVITIES
                  IN USERSPACE1
                  PCTDEACTIVATE 100),
```

```
CONTROL (TABLE CONTROL_DB2ACTIVITIES
         IN USERSPACE1
         PCTDEACTIVATE 100)
AUTOSTART;
```

CREATE EVENT MONITOR (change history)

The CREATE EVENT MONITOR (change history) statement creates an event monitor that can record events for changes to configuration parameters, registry variables, and the execution of DDL statements and utilities.

The event monitor created by the CREATE EVENT MONITOR (change history) statement can also record initial configuration and registry values at event monitor startup time. The set of events recorded depends on the event controls specified in the CREATE EVENT MONITOR statement.

Invocation

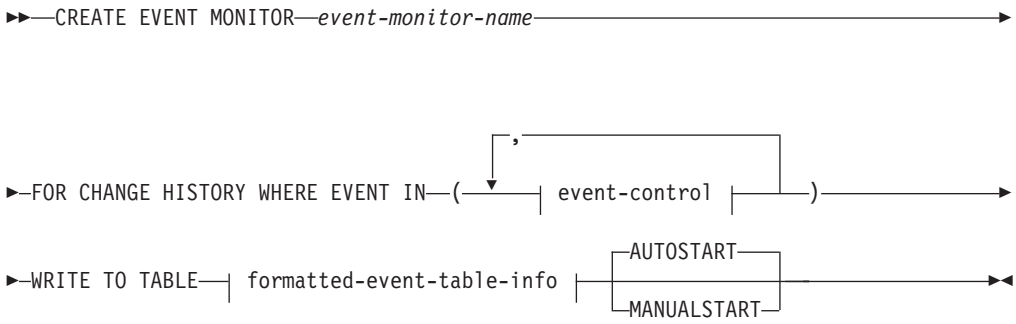
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

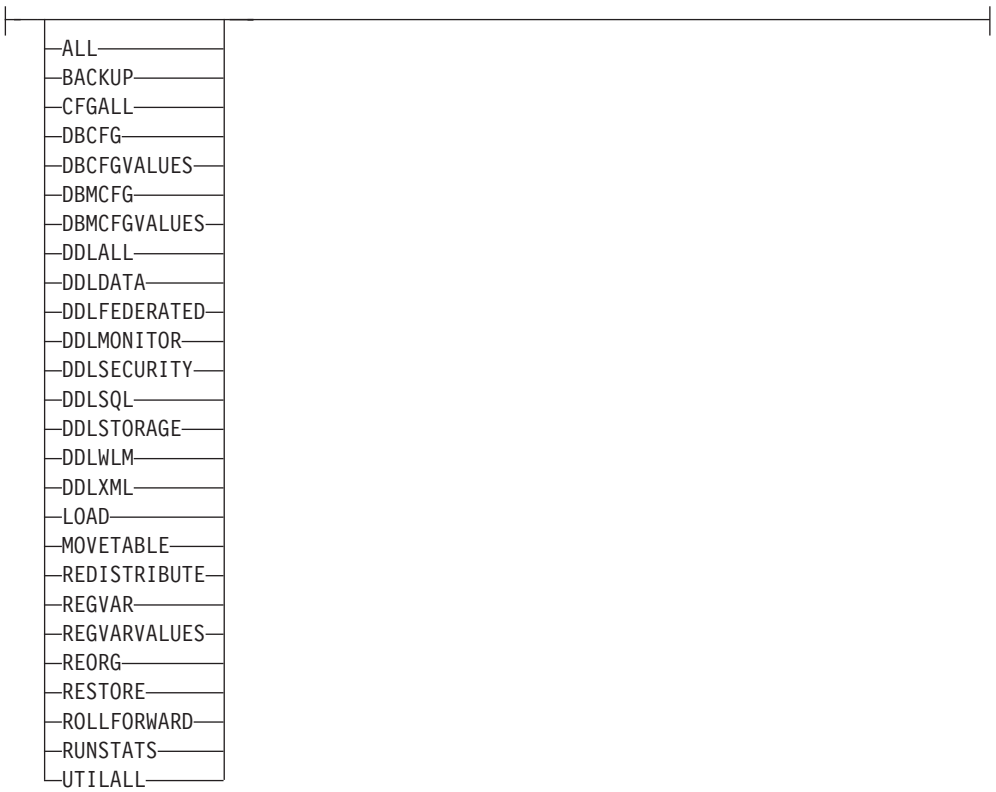
The privileges held by the authorization ID of the statement must include one of the following authorities:

- SQLADM authority
- DBADM authority

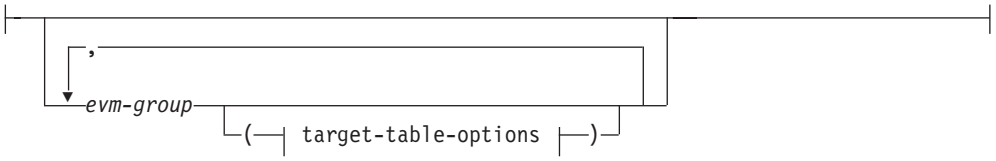
Syntax



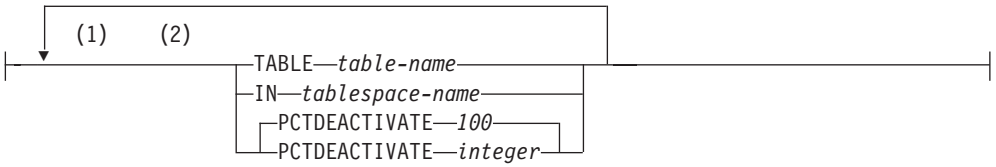
event-control:



formatted-event-table-info:



target-table-options:



Notes:

1 Each condition can be specified only once (SQLSTATE 42613).

2 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

CHANGE HISTORY
Specifies that this event monitor can record events for configuration changes, registry changes, and the execution of DDL statements and utilities. It can also record initial configuration and registry values at event monitor startup time. The set of events recorded depends on the event controls specified in the WHERE EVENT IN clause.

WHERE EVENT IN (event-control, ...)
Specifies one or more event controls used to identify which events are captured by the event monitor.

event-control

ALL Capture all event types.

BACKUP
Capture execution of the online backup utility.

CFGALL
Capture all configuration parameter and registry variable event types.

DBCFG
Capture database configuration parameter changes.

DBCFGVALUES
Record initial values for all database configuration parameters at event monitor startup time if any database configuration parameter update was not captured by the event monitor.

DBMCFG
Capture database manager configuration parameter changes.

DBMCFGVALUES
Record initial values for all database manager configuration parameters at event monitor startup time if any database manager configuration parameter update was not captured by the event monitor.

DDLALL
Capture execution for all types of DDL statements.

DDLDATA
Capture execution of index, sequence, table, and temporary table DDL.

DDLFEDERATED
Capture execution of nickname, server, type mapping, user mapping, and wrapper DDL.

DDLMONITOR
Capture execution of event monitor and usage list DDL.

DDLSECURITY
Capture execution of audit policy, grant, mask, permission role, revoke, security label, security label component, security policy, and trusted context DDL.

DDLSQL
Capture execution of alias, function, method, module, package, procedure, schema, synonym, transform, trigger, type, variable, and view DDL.

DDLSTORAGE
Capture execution of the ALTER DATABASE statement and buffer pool, partition group, storage group, and table space DDL.

DDLWLM
Capture execution of histogram, service class, threshold, work action set, work class set, and workload DDL.

DDLXML
Capture execution of XSROBJECT DDL.

LOAD
Capture execution of the load utility.

MOVETABLE
Capture execution of the table move utility (invocations of the ADMIN_MOVE_TABLE stored procedure).

REDISTRIBUTE
Capture execution of the redistribute partition group utility.

REGVAR
Capture immediate registry variables changes.

REGVARVALUES
Record initial values for registry variables at event monitor startup time.

REORG
Capture execution of the reorg utility.

RESTORE
Capture execution of the online restore utility.

ROLLFORWARD
Capture execution of the online rollforward utility.

RUNSTATS
Capture execution of the runstats utility.

UTILALL
Capture execution of the load, move table, online backup, online restore, online rollforward, redistribute, reorg and runstats utilities.

WRITE TO
Introduces the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of database tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table.

formatted-event-table-info
Defines the target table for a logical data group. Specify this clause for each grouping that is to be recorded. However, if no *evm-group* clauses are specified,

the groups required for the event-control options specified are created along with the CONTROL, CHANGESUMMARY, and EVMONSTART logical groups.

evm-group
Identifies the logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Type of event monitor	evm-group value
Change history	• CONTROL • CHANGESUMMARY • EVMONSTART • TXNCOMPLETION • DDLSTMTEXEC • DBDBMCFG • REGVAR • UTILSTART • UTILSTOP • UTILPHASE • UTILLOCATION

target-table-options
Identifies the target table for the group.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If no name is provided, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
SUBSTRING(evm-group CONCAT '_'
   CONCAT event-monitor-name, 1, 128)
```

IN *tablespace-name*
Defines the table space in which the table is to be created. If no table space name is provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100, where 100 means that the event monitor deactivates when the table space becomes full. The default value is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE parameter to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might deactivate unexpectedly if the table space reaches the threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

If a value for target-table-info is not specified, CREATE EVENT MONITOR processing proceeds as follows:

- A derived table name is used.
- A default table space is chosen.
- The PCTDEACTIVATE parameter defaults to 100.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the database partition on which the event monitor runs is activated. This is the default behavior.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor is activated, it can be deactivated by using the SET EVENT MONITOR STATE statement or by stopping the instance.

Notes

- ***Creation of target event tables:*** The target event tables are created when the CREATE EVENT MONITOR FOR CHANGE HISTORY statement executes if the target tables do not exist.
- ***Previously created event tables:*** During CREATE EVENT MONITOR FOR CHANGE HISTORY processing, if an event table has already been defined for use by another event monitor, the CREATE EVENT MONITOR FOR CHANGE HISTORY statement fails, and an error is returned to the application program. An event table is defined for use by another event monitor if the event table name matches a value found in the SYSCAT.EVENTTABLES catalog view. If the event table exists and is not defined for use by another event monitor, then a table is not created, any other table options parameters are ignored, and processing continues. A warning is returned to the application program.
- ***Dropping event monitors:*** Dropping the event monitor does not drop the event tables. The associated event tables must be manually dropped after the event monitor is dropped.
- ***Pruning:*** The event tables must be manually pruned.
- ***Behavior in a partitioned environment:*** In a partitioned environment, if some target event tables do not exist on a partition, but other target event tables do exist on that same partition, only the data for the target event tables that do exist on that partition is recorded.
- ***FLUSH EVENT MONITOR:*** The FLUSH EVENT MONITOR statement is not applicable to this event monitor and has no effect when issued against it.
- ***Modifying event controls after monitor creation:*** After the change history event monitor is created, the event controls specified using the WHERE EVENT IN clause in the CREATE EVENT MONITOR statement cannot be changed or altered. To change the event controls, the event monitor must be deactivated, dropped, and then recreated specifying a new set of event controls using the WHERE EVENT IN clause.

Examples

- *Example 1*: This example creates a change history event monitor called CFG_WITH_OFFLINE that records configuration changes and initial values for configuration.

```
CREATE EVENT MONITOR CFG_WITH_OFFLINE
   FOR CHANGE HISTORY WHERE EVENT IN (CFGALL)
   WRITE TO TABLE
```

```
    CHANGESUMMARY (TABLE CHG_SUMMARY_HISTORY),
    DBDBMCFG (TABLE DB_DBM_HISTORY),
    REGVAR (TABLE REGVAR_HISTORY)
  AUTOSTART
```

In this example the target tables are explicitly specified. The previous statement creates the following tables:

```
CHG_SUMMARY_HISTORY
DB_DBM_HISTORY
REGVAR_HISTORY
```

- *Example 2*: This example creates a change history event monitor called BKP_REST that collects events describing all online backup and restore utility executions.

```
CREATE EVENT MONITOR BKP_REST
   FOR CHANGE HISTORY WHERE EVENT IN (BACKUP, RESTORE)
   WRITE TO TABLE
```

In this example the target tables are not explicitly specified. The CREATE EVENT MONITOR statement creates only the target tables that are needed based on the controls specified in the WHERE EVENT IN clause, along with tables for the CONTROL, CHANGESUMMARY, and EVMONSTART logical data groups. The BACKUP and RESTORE controls enable collection of utility events for online backup and restore, and require the UTILSTART, UTILSTOP, UTILLOCATION, and UTILPHASE logical data groups. The previous statement creates the following tables:

```
CONTROL_BKP_REST
CHANGESUMMARY_BKP_REST
EVMONSTART_BKP_REST
UTILSTART_BKP_REST
UTILSTOP_BKP_REST
UTILLOCATION_BKP_REST
UTILPHASE_BKP_REST
```

CREATE EVENT MONITOR (locking)

The CREATE EVENT MONITOR (locking) statement creates an event monitor that will record lock-related events that occur when using the database.

Invocation

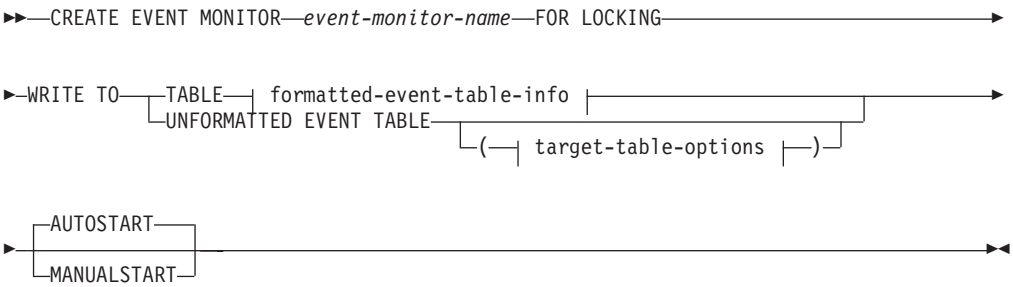
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

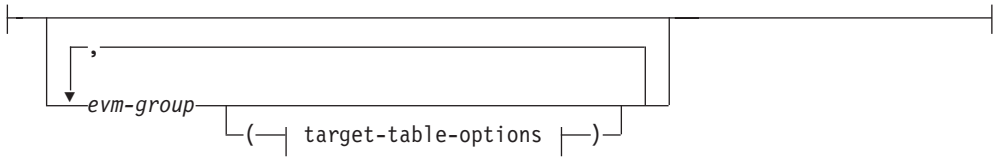
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority

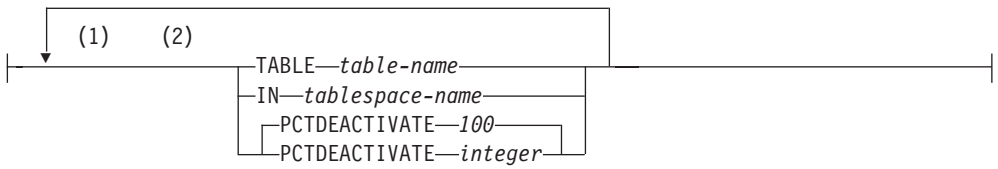
Syntax



formatted-event-table-info:



target-table-options:



Notes:

1 Each table option can be specified a maximum of one time (SQLSTATE 42613).

2 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

LOCKING
Specifies that this passive event monitor will record any lock event produced when DB2 runs into one or more of these conditions:

- LOCKTIMEOUT: the lock has timed-out.
- DEADLOCK: the lock was involved in a deadlock (victim and participant(s)).
- LOCKWAIT: locks that are not acquired in the specified duration.

The creation of the lock event monitor does not indicate that the locking data will be collected immediately. The actual locking event of interest is controlled at the workload level or database level.

WRITE TO
Specifies the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of formatted event tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

formatted-event-table-info
Defines the target formatted event tables for the event monitor. This clause should specify each grouping that is to be recorded. However, if no *evm-group* clauses are specified, all groups for the event monitor type are recorded.

evm-group
Identifies a logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Type of Event Monitor	evm-group Value
Locking	• LOCK • LOCK_PARTICIPANTS • LOCK_PARTICIPANT_ACTIVITIES • LOCK_ACTIVITY_VALUES • CONTROL

UNFORMATTED EVENT TABLE
Specifies that the target for the event monitor is an unformatted event table. The unformatted event table is used to store collected locking event monitor data. Data is stored in an internal binary format within an inlined BLOB column. Each event can insert multiple records into this table and

each inserted record can be of a different type with the associated BLOB content varying as well. The data in the BLOB column is not in a readable format and requires conversion, through use of the **db2evmonfmt** Java-based tool, EVMON_FORMAT_UE_TO_XML table function, or EVMON_FORMAT_UE_TO_TABLES procedure, into a consumable format such as an XML document or a relational table.

target-table-options
Identifies options for the target table. If a value for **target-table-options** is not specified, CREATE EVENT MONITOR FOR LOCKING processing proceeds as follows:

- A derived table name is used (as explained in the description for TABLE *table-name*).
- A default table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.
- PCTDEACTIVATE is set to 100.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If a name is not provided for an unformatted event table, the unqualified name is equal to the *event-monitor-name*, that is, the unformatted event table will be named after the event monitor. If no name is provided for a formatted event table, the unqualified name is derived from *evm-group* and *event-monitorname* as follows:

```
substring(evm-group CONCAT '_'
  CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*
Defines the table space in which the table is to be created. The CREATE EVENT MONITOR FOR LOCKING statement does not create table spaces.

If a table space name is not provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

When specifying the table space name for a formatted event table, the table space's page size affects the INLINE LOB lengths used Consider specifying a table space with as large a page size as possible in order to improve the INSERT performance of the event monitor.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100. The default value is 100, where 100 means the event monitor deactivates when the table space becomes completely full. The default value assumed is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might

deactivate unexpectedly if the table space reaches the threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the database partition on which the event monitor runs is activated. This is the default behavior of the locking event monitor.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor has been activated, it can be deactivated only by using the SET EVENT MONITOR STATE statement or by stopping the instance.

Notes

- The target table is created when the CREATE EVENT MONITOR FOR LOCKING statement executes, if it doesn't already exist.
- During CREATE EVENT MONITOR FOR LOCKING processing, if a table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR FOR LOCKING statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view. If the table exists and is not defined for use by another event monitor, then the event monitor will re-use the table.
- Dropping the event monitor will not drop any tables. Any associated tables must be manually dropped after the event monitor is dropped.
- Lock event data is not automatically pruned from either unformatted event tables or regular tables created by this event monitor. An option for pruning data from UE tables is available when using the EVMON_FORMAT_UE_TO_TABLES procedure. For event monitors that write to regular tables, event data must be pruned manually.
- The FLUSH EVENT MONITOR statement is not applicable to this event monitor and will have no effect when issued against it.
- For unformatted event tables event data is inserted into the table into an inlined BLOB data column. Normally, BLOB data is stored in a separate LOB table space and can experience additional performance overhead as a result. When inlined into the data page of the base table, the BLOB data does not experience this overhead. The DB2 database manager will automatically inline the BLOB data portion of an unformatted event table record if the size of the BLOB data is less than the table space page size minus the record prefix. Therefore to achieve high efficiency and application throughput, it is suggested that you create the event monitor in as large a table space as possible up to and including a 32KB table space and associated bufferpool.

  Example
  The lock event monitor currently has the following two record types:
  – Application Info Record
  – Application Activity Record

  Application Info Record = maximum size 3.5KB

  Application Activity Record = 3KB + SQL statement text size (where SQL statement text size is max 2MB)

The Application Info Record is very small and should always be inlined as long as a 4KB page size is being used. The Application Activity Record will be inlined based on the following formula:

```
Application Activity Record < inline length (Pagesize - overhead non-LOB columns (0.5KB))
   3KB + SQL statement text < inline length (Pagesize - overhead non-LOB columns (0.5KB))

        SQL statement text < Pagesize - nonLOB overhead (1K) - 3KB
        SQL statement text < 16KB - 1KB - 3KB
                          < 12KB
```

Therefore, when using a 16KB pagesize, the lock event monitor records will only be inlined if the SQL statement being captured is less than 12KB in size.

- Create only one locking event monitor per database. Creating more than one locking event monitor uses additional processor cycles and storage, without providing any additional data.

  Important: For compatibility with older versions of the product, all databases are created with the DB2DETAILDEADLOCK event monitor enabled. The locking event monitor introduced in DB2 Version 9.7 is the preferred mechanism for collecting data related to locks; the DB2DETAILEDDEALOCK event monitor is deprecated and might be removed in a future release. When you create a locking event monitor, disable and drop the DB2DETAILEDDEADLOCK event monitor to prevent the collection of duplicate, unnecessary information.

  To remove the DB2DETAILDEADLOCK event monitor, issue the following SQL statements:

  ```
  SET EVENT MONITOR DB2DETAILDEADLOCK state 0
  DROP EVENT MONITOR DB2DETAILDEADLOCK
  ```

- In a partitioned database environment, data is written only to target tables on the database partitions where their table spaces exist. If a table space for a target table does not exist on some database partition, data for that target table is ignored. This behavior allows users to choose a subset of database partitions for monitoring to be chosen, by creating a table space that exists only on certain database partitions.
- In a partitioned database environment, if some target tables do not reside on a database partition, but other target tables do reside on that same database partition, only the data for the target unformatted event tables that do reside on that database partition is recorded.

Examples

- *Example 1:* This example creates a locking event monitor LOCKEVMON that will collect locking events that occur on the database of creation.

  ```
  CREATE EVENT MONITOR LOCKEVMON
    FOR LOCKING
    WRITE TO TABLE
  ```

  This event monitor writes its output to the following tables:

  LOCK_LOCKEVMON

  LOCK_PARTICIPANTS_LOCKEVMON

  LOCK_PARTICIPANT_ACTIVITIES_LOCKEVMON

  LOCK_ACTIVITY_VALUES_LOCKEVMON

  CONTROL_LOCKEVMON

- *Example 2:* This example creates a locking event monitor LOCKEVMON that will collect locking events that occur on the database of creation and store it in the unformatted event table IMRAN.LOCKEVENTS.

```
CREATE EVENT MONITOR LOCKEVMON
  FOR LOCKING
  WRITE TO UNFORMATTED EVENT TABLE (TABLE IMRAN.LOCKEVENTS)
```

- *Example 3:* This example creates a locking event monitor LOCKEVMON that will collect locking events that occur on the database of creation and store it in the unformatted event table IMRAN.LOCKEVENTS in table space APPSPACE. The event monitor will deactivate when the table space becomes 85% full.

```
CREATE EVENT MONITOR LOCKEVMON
  FOR LOCKING
  WRITE TO UNFORMATTED EVENT TABLE
     (TABLE IMRAN.LOCKEVENTS IN APPSPACE PCTDEACTIVATE 85)
```

CREATE EVENT MONITOR (package cache) statement

The CREATE EVENT MONITOR (package cache) statement creates an event monitor that will record events when the cache entry for a section is flushed from the package cache.

Invocation

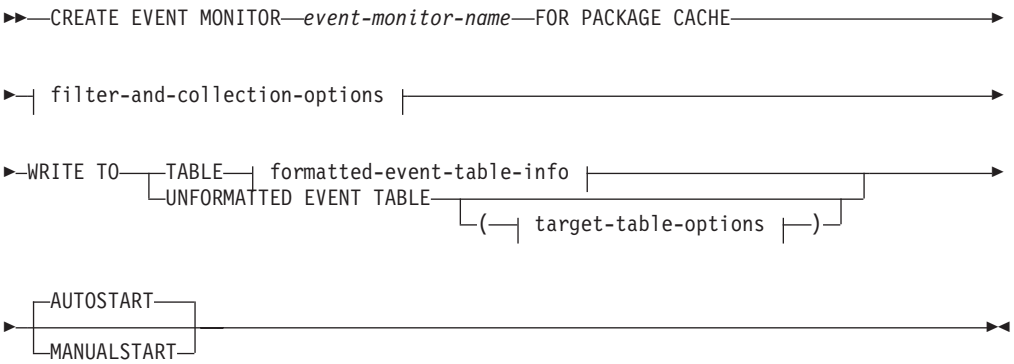
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

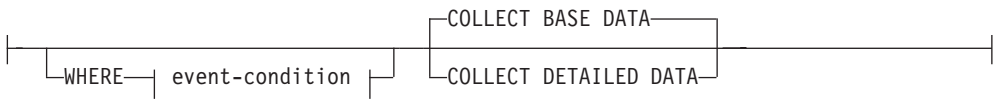
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority

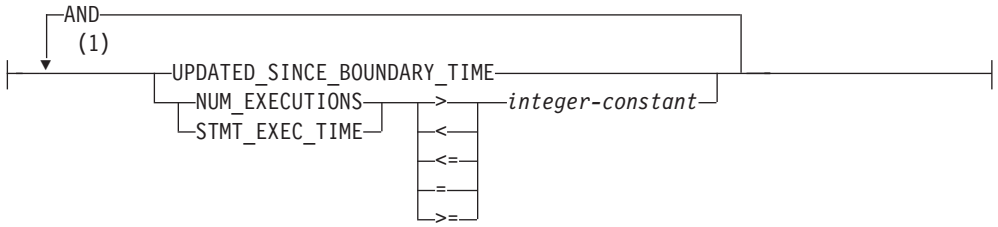
Syntax



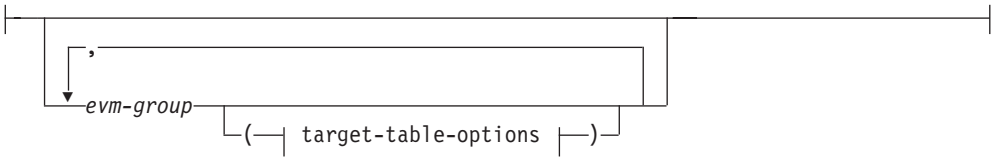
filter-and-collection-options:



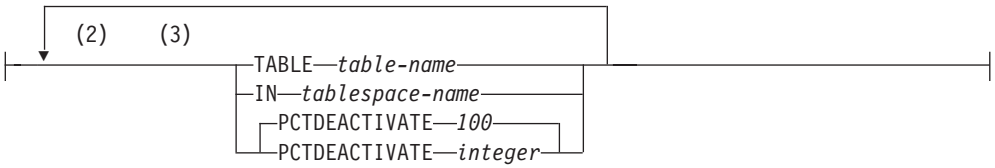
event-condition:



formatted-event-table-info:



target-table-options:



Notes:

1 Each condition can be specified only once (SQLSTATE 42613).

2 Each table option can be specified a maximum of one time (SQLSTATE 42613).

3 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

PACKAGE CACHE
Specifies that this event monitor will record an event when the cache entry for a static or dynamic SQL statement is flushed from the package cache. This event monitor is not passive and will start to record events once it is activated.

filter-and-collection-options
Specify a set of filter and collection options.

WHERE

event-condition
Defines a filter that determines whether entries that are flushed from the package cache should cause an event to occur. If the event condition is TRUE for a particular entry that is being flushed from the package cache, then that entry will be recorded as an event.

This clause is a special form of the WHERE clause that should not be confused with a standard search condition. This is a simple WHERE clause that includes the use of NOT, OR, and LIKE operators, unlike the WHERE clause specified for the CONNECTIONS, TRANSACTIONS, and STATEMENTS event monitors.

If the WHERE clause is not specified, all entries flushed from the package cache will be monitored.

UPDATED_SINCE_BOUNDARY_TIME
Specifies that evicted entries, whose metrics were updated after the boundary time, should be collected by this event monitor. The boundary time is set by calling the MON_GET_PKG_CACHE_STMT table function with the value of the input key "updated_boundary_time" set as the name of this event monitor.

The boundary time is initially set to the activation timestamp of the event monitor.

NUM_EXECUTIONS > | < | <= | = | >= *integer-constant*
Specifies that the monitor element **num_executions** should be compared with the *integer-constant* in order to determine whether to generate an event. NUM_EXECUTIONS is the number of times that the section of the evicted entry was executed.

Note: The **num_executions** monitor element counts all executions of a statement, whether or not the execution of the statement contributed to the activity metrics that are reported.

STMT_EXEC_TIME > | < | <= | = | >= *integer-constant*
Specifies that the monitor element **stmt_exec_time** should be compared with the *integer-constant* in order to determine whether to generate an event. STMT_EXEC_TIME is the total aggregated time spent executing the statement of the evicted entry. The unit of time for the *integer-constant* must be specified as milliseconds.

COLLECT BASE DATA
Specifies that the same level of information returned by the MON_GET_PKG_CACHE_STMT table function should be captured. This is the default collect option.

COLLECT DETAILED DATA
Specifies that the BASE level information should be collected as well as the runtime executable section of the flushed entry.

WRITE TO
Specifies the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of database tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

formatted-event-table-info
Defines the target formatted event tables for the event monitor. This clause should specify each grouping that is to be recorded. However, if no *evm-group* clauses are specified, all groups for the event monitor type are recorded.

evm-group
Identifies a logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Type of Event Monitor	evm-group Value
Package Cache	• PKGCACHE • PKGCACHE_METRICS • PKGCACHE_STMT_ARGS • CONTROL

UNFORMATTED EVENT TABLE
Specifies that the target for the event monitor is an unformatted event table. The unformatted event table is used to store collected package cache event monitor data. Data is stored in its original binary format within an inlined BLOB column. The BLOB column can contain multiple binary records of different types. The data in the BLOB column is not in a readable format and requires conversion, through use of the **db2evmonfmt** Java-based tool, EVMON_FORMAT_UE_TO_XML table function, or EVMON_FORMAT_UE_TO_TABLES procedure, into a consumable format such as an XML document or a relational table.

target-table-options
Identifies options for the target table. If a value for **target-table-options** is not specified, CREATE EVENT MONITOR FOR PACKAGE CACHE processing proceeds as follows:

- A derived table name is used (as explained in the description for TABLE *table-name*).
- A default table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.
- PCTDEACTIVATE is set to 100.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If a name is not provided for an unformatted event table, the unqualified name is equal to the *event-monitor-name*, that is, the unformatted event table will be named after the event monitor. If no name is provided for a formatted event table, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
substring(evm-group CONCAT '_'
  CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*
Specifies the table space in which the table is to be created. The CREATE EVENT MONITOR FOR PACKAGE CACHE statement does not create table spaces.

If a table space name is not provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

The table space's page size affects the INLINE LOB lengths used. Consider specifying a table space with as large a page size as possible in order to improve the INSERT performance of the event monitor.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor

automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100, where 100 means that the event monitor deactivates when the table space becomes completely full. The default value assumed is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE parameter to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might deactivate unexpectedly if the table space reaches the threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the database partition on which the event monitor runs is activated. This is the default behavior of the package cache event monitor.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor has been activated, it can be deactivated by using the SET EVENT MONITOR STATE statement or by stopping the instance.

Notes

- The target table is created when the CREATE EVENT MONITOR FOR PACKAGE CACHE statement executes, if it doesn't already exist.
- During CREATE EVENT MONITOR FOR PACKAGE CACHE processing, if a table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR FOR PACKAGE CACHE statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view. If the table exists and is not defined for use by another event monitor, then the event monitor will re-use the table.
- Dropping the event monitor will not drop any tables. Any associated tables must be manually dropped after the event monitor is dropped.
- Lock event data is not automatically pruned from either unformatted event tables or regular tables created by this event monitor. An option for pruning data from UE tables is available when using the EVMON_FORMAT_UE_TO_TABLES procedure. For event monitors that write to regular tables, event data must be pruned manually.
- In a partitioned database environment, data is written to target tables only on the members where their table spaces exist. If a table space for a target table does not exist on a member, the event data that would be written to that target table is not captured on that member. This behavior allows users to choose a subset of members for monitoring, by creating a table space that exists only on certain members.
- In a partitioned database environment, data is written to target tables only on the member where the entries are evicted from the database package cache.
- In a partitioned database environment, if some target tables do not reside on a member, but other target tables do reside on that same member, only the data for the target unformatted event tables that do reside on that member is recorded.
- The FLUSH EVENT MONITOR statement is not applicable to this event monitor and will have no effect when issued against it.

- After the package cache event monitor is created, the filter and control options cannot be changed or altered. To change the filter and control options, the event monitor must be deactivated, dropped, and then recreated with the new filter and control options.

Use large table space for high throughput
Event data is inserted into the unformatted event table into an inlined BLOB data column. Normally, BLOB data is stored in a separate LOB table space and can experience additional performance overhead as a result. When inlined into the data page of the base table, the BLOB data does not experience this overhead. The DB2 database manager will automatically inline the BLOB data portion of an unformatted event table record if the size of the BLOB data is less than the table space page size minus the record prefix. Therefore, to achieve high efficiency and application throughput, it is suggested that you create the event monitor in as large a table space as possible, up to and including a 32 KB table space, and associated bufferpool.

Inline of package cache records
For the package cache event monitor, the size of the **stmt_text**, **comp_env_desc**, and the **section_env** monitor elements will determine if the package cache record will be inlined or not. If the total of these fields exceeds the table space size, then the record will not be inlined.

Determine if EVENT_DATA is inlined
Use the ADMIN_IS_INLINED and ADMIN_EST_INLINE_LENGTH functions to determine whether the record is inlined and get an estimate of the inline length that is required.

Restrictions

- During database deactivation, evicted entries will not be collected by the package cache event monitor.

Examples

- *Example 1:* This example creates a package cache event monitor called CACHEEVMON that will collect data related to package cache section eviction events and write the data to tables.

```
CREATE EVENT MONITOR CACHEEVMON
  FOR PACKAGE CACHE
  WRITE TO TABLE
```

  This event monitor writes its output to the following tables:

  PKGCACHE_CACHEEVMON

  PKGCACHE_METRICS_CACHEEVMON

  PKGCACHE_STMT_ARGS

  CONTROL_CACHEEVMON

- *Example 2:* This example creates a package cache event monitor called CACHESTMTEVMON that will collect data related to package cache section eviction events and store it in the unformatted event table ALAN.STMTEVENTS.

```
CREATE EVENT MONITOR CACHESTMTEVMON
  FOR PACKAGE CACHE
  WRITE TO UNFORMATTED EVENT TABLE (TABLE ALAN.STMTEVENTS)
```

- *Example 3:* This example creates a package cache event monitor called CACHESTMTEVMON that will collect data related to package cache section

eviction events and store it in the unformatted event table ALAN.STMTEVENTS in table space APPSPACE. The event monitor will deactivate when the table space becomes 85% full.

```
CREATE EVENT MONITOR CACHESTMTEVMON
  FOR PACKAGE CACHE
  WRITE TO UNFORMATTED EVENT TABLE
     (TABLE ALAN.STMTEVENTS IN APPSPACE PCTDEACTIVATE 85)
```

CREATE EVENT MONITOR (statistics)

The CREATE EVENT MONITOR (statistics) statement defines a monitor that will record statistics events that occur when using the database. The definition of the statistics event monitor also specifies where the database should record the events.

Invocation

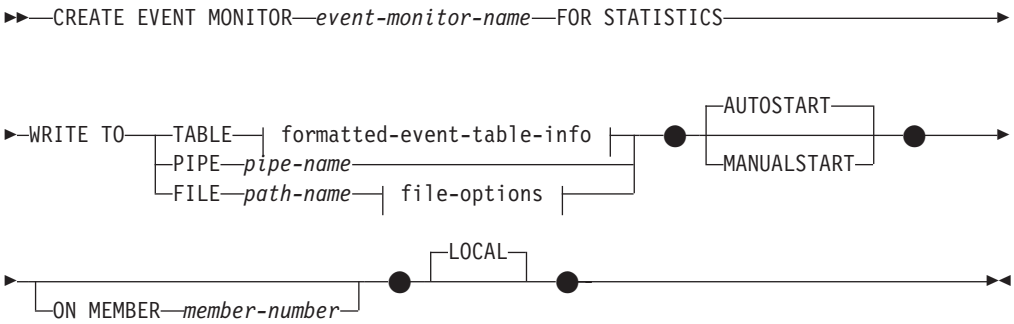
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

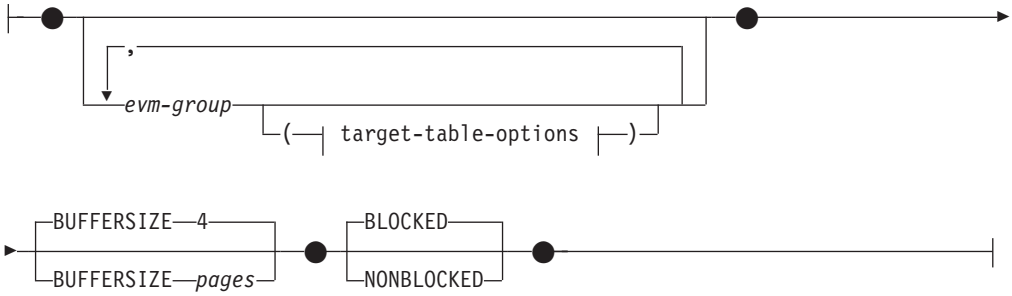
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority
- WLMADM authority

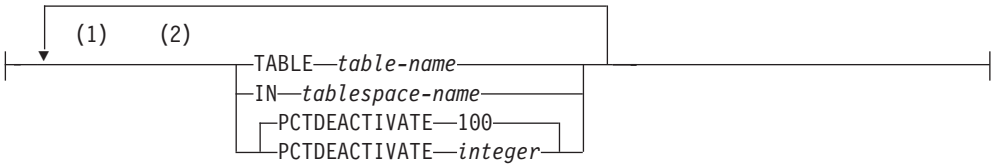
Syntax



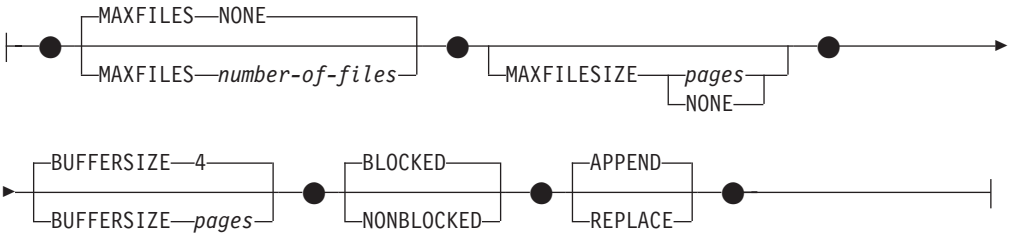
formatted-event-table-info:



target-table-options:



file-options:



Notes:

1 Each clause can be specified only once.

2 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

STATISTICS
Specifies that the event monitor records a service class, workload, or work class event:

- Every *period* minutes, where *period* is the value of the **wlm_collect_int** database configuration parameter
- When the wlm_collect_stats procedure is called

WRITE TO
Introduces the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of database tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

formatted-event-table-info
Defines the target tables for an event monitor. This clause should be

specified for each grouping that is to be recorded. However, if no evm-group-info clauses are specified, all groups for the event monitor type are recorded.

evm-group
Identifies the logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Type of Event Monitor	evm-group Value
Statistics	• QSTATS • SCSTATS • SCMETRICS • WCSTATS • WLSTATS • WLMETRICS • HISTOGRAMBIN • CONTROL

target-table-options
Identifies the target table for the group.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If no name is provided, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
substring(evm-group CONCAT '_'
  CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*
Defines the table space in which the table is to be created. If no table space name is provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100, where 100 means that the event monitor deactivates when the table space becomes completely full. The default value assumed is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE parameter to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might deactivate unexpectedly if the table space reaches the

threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

If a value for *target-table-options* is not specified, CREATE EVENT MONITOR processing proceeds as follows:

- A derived table name is used.
- A default table space is chosen.
- PCTDEACTIVATE defaults to 100.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K pages). Table event monitors insert all data from a buffer, and issues a COMMIT once the buffer has been processed. The larger the buffers, the larger the commit scope used by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event monitors. When a monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O.

The default size of each buffer is 4 pages (two 16K buffers are allocated). The minimum size is 1 page. The maximum size of the buffers is limited by the size of the monitor heap, because the buffers are allocated from that heap. If many event monitors are being used at the same time, increase the size of the **mon_heap_sz** database manager configuration parameter.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

NONBLOCKED
Specifies that each agent that generates an event should not wait for the event buffer to be written out to disk if the agent determines that both event buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of BLOCKED event monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems.

PIPE
Specifies that the target for the event monitor data is a named pipe. The event monitor writes the data to the pipe in a single stream (that is, as if it were a single, infinitely long file). When writing the data to a pipe, an event monitor does not perform blocked writes. If there is no room in the pipe buffer, then the event monitor will discard the data. It is the monitoring application's responsibility to read the data promptly if it wishes to ensure no data loss.

pipe-name
The name of the pipe (FIFO on AIX) to which the event monitor will write the data.

The naming rules for pipes are platform specific. On UNIX operating systems, pipe names are treated like file names. As a result, relative pipe names are permitted, and are treated like relative path-names (refer to the description for *path-name*). On Windows, however, there is a special syntax for a pipe name and, as a result, absolute pipe names are required.

The existence of the pipe will not be checked at event monitor creation time. It is the responsibility of the monitoring application to have created and opened the pipe for reading at the time that the event monitor is activated. If the pipe is not available at this time, then the event monitor will turn itself off, and will log an error. (That is, if the event monitor was activated at database start time as a result of the AUTOSTART option, then the event monitor will log an error in the system error log.) If the event monitor is activated via the SET EVENT MONITOR STATE SQL statement, then that statement will fail (SQLSTATE 58030).

FILE

Indicates that the target for the event monitor data is a file (or set of files). The event monitor writes out the stream of data as a series of 8 character numbered files, with the extension "evt". (for example, 00000000.evt, 00000001.evt, and 00000002.evt). The data should be considered to be one logical file even though the data is broken up into smaller pieces (that is, the start of the data stream is the first byte in the file 00000000.evt; the end of the data stream is the last byte in the file nnnnnnnn.evt).

The maximum size of each file can be defined as well as the maximum number of files. An event monitor will never split a single event record across two files. However, an event monitor may write related records in two different files. It is the responsibility of the application that uses this data to keep track of such related information when processing the event files.

path-name

The name of the directory in which the event monitor should write the event files data. The path must be known at the server; however, the path itself could reside on another database partition (for example, on a UNIX system, this might be an NFS mounted file). A string constant must be used when specifying the *path-name*.

The directory does not have to exist at CREATE EVENT MONITOR time. However, a check is made for the existence of the target path when the event monitor is activated. At that time, if the target path does not exist, an error (SQLSTATE 428A3) is raised.

If an absolute path (a path that starts with the root directory on AIX, or a disk identifier on Windows) is specified, the specified path will be the one used. In environments other than DB2 pureScale, if a relative path (a path that does not start with the root) is specified, then the path relative to the DB2EVENT directory in the database directory will be used. In a DB2 pureScale environment, if a relative path is specified, then the path relative to the database owning directory in the database directory will be used.

It is possible to specify two or more event monitors that have the same target path. However, once one of the event monitors has been activated for the first time, and as long as the target directory is not empty, it will be impossible to activate any of the other event monitors.

file-options

Specifies the options for the file format.

MAXFILES NONE

Specifies that there is no limit to the number of event files that the event monitor will create. This is the default.

MAXFILES *number-of-files*
Specifies that there is a limit on the number of event monitor files that will exist for a particular event monitor at any time. Whenever an event monitor has to create another file, it will check to make sure that the number of .evt files in the directory is less than *number-of-files*. If this limit has already been reached, then the event monitor will turn itself off.

If an application removes the event files from the directory after they have been written, then the total number of files that an event monitor can produce can exceed *number-of-files*. This option has been provided to allow a user to guarantee that the event data will not consume more than a specified amount of disk space.

MAXFILESIZE *pages*
Specifies that there is a limit to the size of each event monitor file. Whenever an event monitor writes a new event record to a file, it checks that the file will not grow to be greater than *pages* (in units of 4K pages). If the resulting file would be too large, then the event monitor switches to the next file. The default for this option is:

- Windows - 200 4K pages
- UNIX - 1000 4K pages

The number of pages must be greater than at least the size of the event buffer in pages. If this requirement is not met, then an error (SQLSTATE 428A4) is raised.

MAXFILESIZE NONE
Specifies that there is no set limit on a file's size. If MAXFILESIZE NONE is specified, then MAXFILES 1 must also be specified. This option means that one file will contain all of the event data for a particular event monitor. In this case the only event file will be 00000000.evt.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K pages). All event monitor file I/O is buffered to improve the performance of the event monitors. The larger the buffers, the less I/O will be performed by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event monitors. When the monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O.

The default size of each buffer is 4 pages (two 16K buffers are allocated). The minimum size is 1 page. The maximum size of the buffers is limited by the value of the MAXFILESIZE parameter, as well as the size of the monitor heap, because the buffers are allocated from that heap. If many event monitors are being used at the same time, increase the size of the **mon_heap_sz** database manager configuration parameter.

Event monitors that write their data to a pipe also have two internal (non-configurable) buffers that are each 1 page in size. These buffers are also allocated from the monitor heap (MON_HEAP). For each active event monitor that has a pipe target, increase the size of the database heap by 2 pages.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

NONBLOCKED
Specifies that each agent that generates an event should not wait for the event buffer to be written out to disk if the agent determines that both event buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of BLOCKED event monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems.

APPEND
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will append the new event data to the existing stream of data files. When the event monitor is reactivated, it will resume writing to the event files as if it had never been turned off. APPEND is the default option.

The APPEND option does not apply at CREATE EVENT MONITOR time, if there is existing event data in the directory where the newly created event monitor is to write its event data.

REPLACE
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will erase all of the event files and start writing data to file 00000000.evt.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor has been activated, it can be deactivated only by using the SET EVENT MONITOR STATE statement or by stopping the instance.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the member on which the event monitor runs is activated. This is the default behavior of the statistics event monitor.

ON MEMBER *member-number*
Specifies the member on which a file or pipe event monitor is to run. When the monitoring scope is defined as LOCAL, data is collected only on the specified member. The I/O component will physically run on the specified member, writing records to the specified file or pipe. When DB2 pureScale feature is enabled, -1 is the default. If -1 is specified, it allows the I/O component to run from any active member. Additionally, in the event that the I/O component is no longer able to run on a given member, the event monitor will be restarted with the I/O component running on another available active member.

This clause is not valid for table event monitors. In a partitioned database environment, write-to-table event monitors will run and write events on all database partitions where table spaces for target tables are defined. In a DB2 pureScale environment, write-to-table event monitors will record events on all active members.

If this clause is not specified and the DB2 pureScale feature is not enabled, the currently connected database partition number (for the application) is used. If

this clause is not specified and DB2 pureScale is enabled, the I/O component is able to run on any currently connected member.

LOCAL
The event monitor reports only on the member that is running. It gives a partial trace of the database activity. This is the default.

This clause is valid for file or pipe monitors. It is not valid for table event monitors.

GLOBAL is not a valid scope for this type of event monitor.

Rules

- The STATISTICS event type cannot be combined with any other event types in a particular event monitor definition.

Notes

- Event monitor definitions are recorded in the SYSCAT.EVENTMONITORS catalog view. The events themselves are recorded in the SYSCAT.EVENTS catalog view. The names of target tables are recorded in the SYSCAT.EVENTTABLES catalog view.
- If the member on which the event monitor is to run is not active, event monitor activation occurs when that member next activates.
- After an event monitor is activated, it behaves like an autostart event monitor until that event monitor is explicitly deactivated or the instance is recycled. That is, if an event monitor is active when a member is deactivated, and that member is subsequently reactivated, the event monitor is also explicitly reactivated.
- If you create the event monitor such that the logical data groups event_scstats or event_wlstats are included in the event monitor output, metrics are reported in two XML documents contained in the event monitor output. The metrics elements contained in the `metrics` document are reset to 0 after each monitoring interval. The metrics contained in `details_xml` are not reset after each interval; instead, they continue to accumulate until the next database activation.

  Important: Starting with Version 10.1 Fix Pack 1, the XML document details_xml is deprecated in the statistics event monitor, and might be removed in a future release. For more information, see Reporting of metrics in details_xml by the statistics event monitor has been deprecated"Reporting of metrics in details_xml by the statistics event monitor has been deprecated" in *What's New for DB2 Version 10.1*.
- ***Write to table event monitors***: General notes:
  - All target tables are created when the CREATE EVENT MONITOR statement executes.
  - If the creation of a table fails for any reason, an error is passed back to the application program, and the CREATE EVENT MONITOR statement fails.
  - A target table can only be used by one event monitor. During CREATE EVENT MONITOR processing, if a target table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view.
  - During CREATE EVENT MONITOR processing, if a table already exists, but is *not* defined for use by another event monitor, no table is created, and processing continues. A warning is passed back to the application program.

– Any table spaces must exist before the CREATE EVENT MONITOR statement is executed. The CREATE EVENT MONITOR statement does not create table spaces.
– If specified, the LOCAL and GLOBAL keywords are ignored. With WRITE TO TABLE event monitors, an event monitor output process or thread is started on each member in the instance, and each of these processes reports data only for the member on which it is running.
– The following event types from the flat monitor log file or pipe format are not recorded by write to table event monitors:
  - LOG_STREAM_HEADER
  - LOG_HEADER
  - DB_HEADER (Elements db_name and db_path are not recorded. The element conn_time is recorded in CONTROL.)
– In a partitioned database environment, data is only written to target tables on the database partitions where their table spaces exist. If a table space for a target table does not exist on some database partition, data for that target table is ignored. This behavior allows users to choose a subset of member for monitoring, by creating a table space that exists only on certain member. In a DB2 pureScale environment, data will be written from every member.

  In a partitioned database environment, if some target tables do not reside on a database partition, but other target tables do reside on that same database partition, only the data for the target tables that do reside on that database partition is recorded.
– Users must manually prune all target tables.

Table Columns:

– Column names in a table match an event monitor element identifier. Any event monitor element that does not have a corresponding target table column is ignored.
– Use the db2evtbl command to build a CREATE EVENT MONITOR command that includes a complete list of elements for a group.
– The types of columns being used for monitor elements correlate to the following mapping:

```
SQLM_TYPE_STRING                    CHAR[n], VARCHAR[n] or CLOB(n)
                                    (If the data in the event monitor
                                    record exceeds n bytes,
                                    it is truncated.)
SQLM_TYPE_U8BIT and SQLM_TYPE_8BIT  SMALLINT, INTEGER or BIGINT
SQLM_TYPE_16BIT and SQLM_TYPE_U16BIT SMALLINT, INTEGER or BIGINT
SQLM_TYPE_32BIT and SQLM_TYPE_U32BIT INTEGER or BIGINT
SQLM_TYPE_U64BIT and SQLM_TYPE_64BIT BIGINT
sqlm_timestamp                      TIMESTAMP
sqlm_time(elapsed time)             BIGINT
sqlca:
  sqlerrmc                          VARCHAR[72]
  sqlstate                          CHAR[5]
  sqlwarn                           CHAR[11]
  other fields                      INTEGER or BIGINT
```

– Columns are defined to be NOT NULL.
– Unlike other target tables, the columns in the CONTROL table do not match monitor element identifiers. Columns are defined as follows:

```
Column Name      Data Type     Nullable  Description
-----------      ---------     --------  -----------
PARTITION_KEY    INTEGER       N         Distribution key (partitioned
                                         database only)
PARTITION_NUMBER INTEGER       N         Database partition number
```

```
                                                    (partitioned database only)
EVMONNAME        VARCHAR(128)  N          Name of the event monitor
MESSAGE          VARCHAR(128)  N          Describes the nature of
                                          the MESSAGE_TIME column.
                                          This can be one of
                                          the following values:
                                          - FIRST_CONNECT (the time
                                            of the first connect to the
                                            database after activation)
                                          - EVMON_START (the time that
                                            the event monitor listed
                                            in EVMONNAME was started)
                                          - OVERFLOWS:n (denotes that
                                            n records were discarded
                                            because of buffer overflow)
                                          - LAST_DROPPED_RECORD
                                            (the last time that
                                            an overflow occurred)
MESSAGE_TIME   TIMESTAMP       N          Timestamp
```

– In a partitioned database environment, the first column of each table is named PARTITION_KEY, is NOT NULL, and is of type INTEGER. This column is used as the distribution key for the table. The value of this column is chosen so that each event monitor process inserts data into the member on which the process is running; that is, insert operations are performed locally on the member where the event monitor process is running. On any database partition, the PARTITION_KEY field will contain the same value. This means that if a database partition is dropped and data redistribution is performed, all data on the dropped database partition will go to one other database partition instead of being evenly distributed. Therefore, before removing a database partition, consider deleting all table rows on that database partition.
– In a partitioned database environment, a column named PARTITION_NUMBER can be defined for each table. This column is NOT NULL and is of type INTEGER. It contains the number of the database partition on which the data was inserted. Unlike the PARTITION_KEY column, the PARTITION_NUMBER column is not mandatory. The PARTITION_NUMBER column is not allowed in a non-partitioned database environment.

Table Attributes:

– Default table attributes are used. Besides distribution key (partitioned databases only), no extra options are specified when creating tables.
– Indexes on the table can be created.
– Extra table attributes (such as volatile, RI, triggers, constraints, and so on) can be added, but the event monitor process (or thread) will ignore them.
– If "not logged initially" is added as a table attribute, it is turned off at the first COMMIT, and is not set back on.

Event Monitor Activation:

– When an event monitor activates, all target table names are retrieved from the SYSCAT.EVENTTABLES catalog view.
– In a partitioned database environment, activation processing occurs on every member of the instance. On a particular member, activation processing determines the table spaces and database partition groups for each target table. The event monitor only activates on a member if at least one target table exists on that database partition. Moreover, if some target table is not found on a database partition, that target table is flagged so that data destined for that table is dropped during runtime processing.

- If a target table does not exist when the event monitor activates (or, in a partitioned database environment, if the table space does not reside on a database partition), activation continues, and data that would otherwise be inserted into this table is ignored.
- Activation processing validates each target table. If validation fails, activation of the event monitor fails, and messages are written to the administration log.
- During activation in a partitioned database environment, the CONTROL table rows for FIRST_CONNECT and EVMON_START are only inserted on the catalog database partition. This requires that the table space for the control table exist on the catalog database partition. If it does not exist on the catalog database partition, these inserts are not performed.
- In a partitioned database environment, if a member is not yet active when a write to table event monitor is activated, the event monitor will be activated the next time that member is activated.

Run Time:

- An event monitor runs with DATAACCESS authority.
- If, while an event monitor is active, an insert operation into a target table fails:
  - Uncommitted changes are rolled back.
  - A message is written to the administration log.
  - The event monitor is deactivated.
- If an event monitor is active, it performs a local COMMIT when it has finished processing an event monitor buffer.
- In an environment other than a partitioned database or a DB2 pureScale environment, all write to table event monitors are deactivated when the last application terminates (and the database has not been explicitly activated). In a DB2 pureScale environment, write to table event monitors are deactivated on a given member when the member stops and is reactivated when the member restarts. In a partitioned database environment, write to table event monitors are deactivated when the catalog partition deactivates.
- The DROP EVENT MONITOR statement does not drop target tables.
- Whenever a write-to-table event monitor activates, it will acquire IN table locks on each target table in order to prevent them from being modified while the event monitor is active. Table locks are maintained on all tables while the event monitor is active. If exclusive access is required on any of the target tables (for example, when a utility is to be run), first deactivate the event monitor to release the table locks before attempting such access.

- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DBPARTITIONNUM or NODE can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - Commas can be used to separate multiple options in the *target-table-options* clause

Example

Define a statistics event monitor named DB2STATISTICS

```
CREATE EVENT MONITOR DB2STATISTICS
    FOR STATISTICS
    WRITE TO TABLE
    SCSTATS (TABLE SCSTATS_DB2STATISTICS
```

```
         IN USERSPACE1
         PCTDEACTIVATE 100),
WCSTATS (TABLE WCSTATS_DB2STATISTICS
         IN USERSPACE1
         PCTDEACTIVATE 100),
WLSTATS (TABLE WLSTATS_DB2STATISTICS
         IN USERSPACE1
         PCTDEACTIVATE 100),
QSTATS  (TABLE QSTATS_DB2STATISTICS
         IN USERSPACE1
         PCTDEACTIVATE 100),
HISTOGRAMBIN (TABLE HISTOGRAMBIN_DB2STATISTICS
              IN USERSPACE1
              PCTDEACTIVATE 100),
CONTROL (TABLE CONTROL_DB2STATISTICS
         IN USERSPACE1
         PCTDEACTIVATE 100)
AUTOSTART;
```

CREATE EVENT MONITOR (threshold violations)

The CREATE EVENT MONITOR (threshold violations) statement defines a monitor that will record threshold violation events that occur when using the database. The definition of the threshold violations event monitor also specifies where the database should record the events.

Invocation

Staring in Version 10 Fix Pack 1 and later fix packs, the threshold violations event monitor can collect more information about the application that violated the threshold. The addition of these monitor elements does not affect existing threshold violations event monitors, but in order to collect the additional application information existing monitors must dropped and recreated.

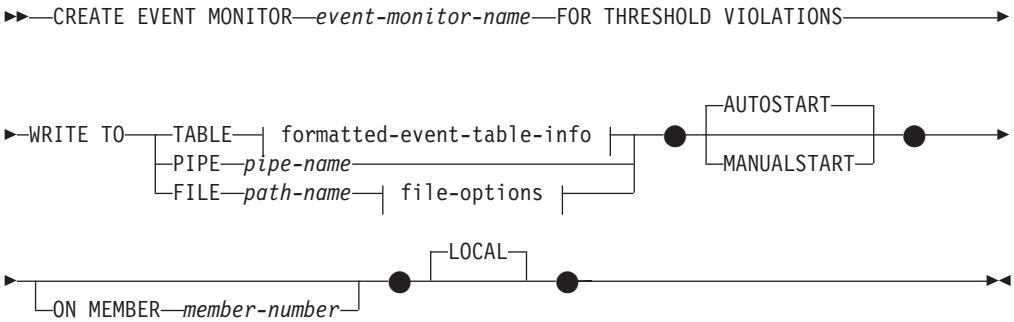
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

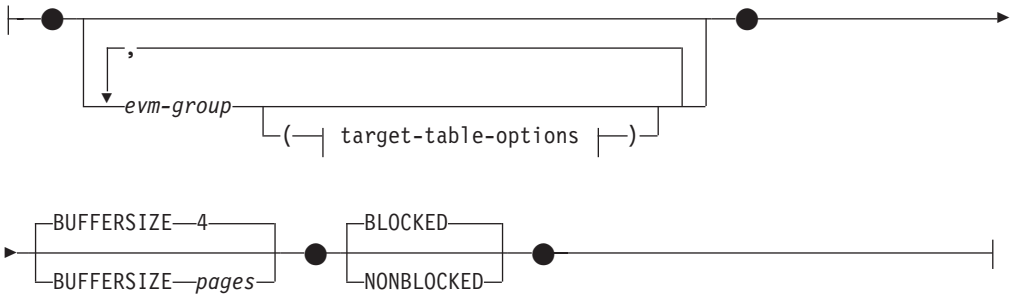
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority
- WLMADM authority

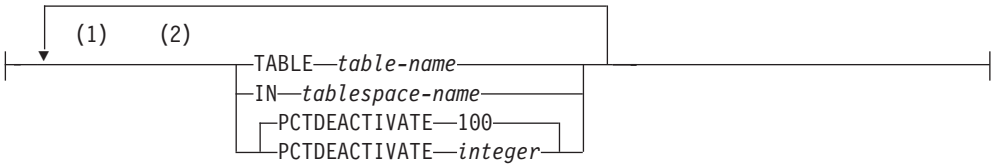
Syntax



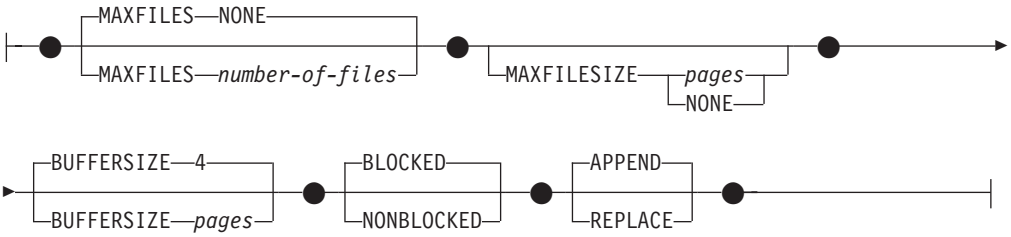
formatted-event-table-info:



target-table-options:



file-options:



Notes:

1 Each clause can be specified only once.

2 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

THRESHOLD VIOLATIONS
Specifies that the event monitor records a threshold violation event when a threshold is violated. Such events can be recorded at any point in the life of an activity, not just at completion.

WRITE TO
Introduces the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of database tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

formatted-event-table-info
Defines the target tables for an event monitor. This clause should be specified for each grouping that is to be recorded. However, if no evm-group-info clauses are specified, all groups for the event monitor type are recorded.

evm-group
Identifies the logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Type of Event Monitor	evm-group Value
Threshold violations	• THRESHOLDVIOLATIONS • CONTROL

target-table-options
Identifies the target table for the group.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. If no name is provided, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
substring(evm-group CONCAT '_'
  CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*
Defines the table space in which the table is to be created. If no table space name is provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100, where 100 means that the event monitor deactivates when the table space becomes completely full. The default value assumed is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE parameter to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might deactivate unexpectedly if the table space reaches the threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

If a value for *target-table-options* is not specified, CREATE EVENT MONITOR processing proceeds as follows:

- A derived table name is used.
- A default table space is chosen.
- The PCTDEACTIVATE parameter defaults to 100.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K

pages). Table event monitors insert all data from a buffer, and issues a COMMIT once the buffer has been processed. The larger the buffers, the larger the commit scope used by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event monitors. When a monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O.

The default size of each buffer is 4 pages (two 16K buffers are allocated). The minimum size is 1 page. The maximum size of the buffers is limited by the size of the monitor heap, because the buffers are allocated from that heap. If many event monitors are being used at the same time, increase the size of the **mon_heap_sz** database manager configuration parameter.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

NONBLOCKED
Specifies that each agent that generates an event should not wait for the event buffer to be written out to disk if the agent determines that both event buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of BLOCKED event monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems.

PIPE
Specifies that the target for the event monitor data is a named pipe. The event monitor writes the data to the pipe in a single stream (that is, as if it were a single, infinitely long file). When writing the data to a pipe, an event monitor does not perform blocked writes. If there is no room in the pipe buffer, then the event monitor will discard the data. It is the monitoring application's responsibility to read the data promptly if it wishes to ensure no data loss.

pipe-name
The name of the pipe (FIFO on AIX) to which the event monitor will write the data.

The naming rules for pipes are platform specific. On UNIX operating systems, pipe names are treated like file names. As a result, relative pipe names are permitted, and are treated like relative path-names (refer to the description for *path-name*). On Windows, however, there is a special syntax for a pipe name and, as a result, absolute pipe names are required.

The existence of the pipe will not be checked at event monitor creation time. It is the responsibility of the monitoring application to have created and opened the pipe for reading at the time that the event monitor is activated. If the pipe is not available at this time, then the event monitor will turn itself off, and will log an error. (That is, if the event monitor was activated at database start time as a result of the AUTOSTART option, then the event monitor will log an error in the system error log.) If the event monitor is activated via the SET EVENT MONITOR STATE SQL statement, then that statement will fail (SQLSTATE 58030).

FILE
Indicates that the target for the event monitor data is a file (or set of files). The event monitor writes out the stream of data as a series of 8 character numbered files, with the extension "evt". (for example, 00000000.evt, 00000001.evt, and 00000002.evt). The data should be considered to be one logical file even though the data is broken up into smaller pieces (that is, the start of the data stream is the first byte in the file 00000000.evt; the end of the data stream is the last byte in the file nnnnnnnn.evt).

The maximum size of each file can be defined as well as the maximum number of files. An event monitor will never split a single event record across two files. However, an event monitor may write related records in two different files. It is the responsibility of the application that uses this data to keep track of such related information when processing the event files.

path-name
The name of the directory in which the event monitor should write the event files data. The path must be known at the server; however, the path itself could reside on another database partition (for example, on a UNIX system, this might be an NFS mounted file). A string constant must be used when specifying the *path-name*.

The directory does not have to exist at CREATE EVENT MONITOR time. However, a check is made for the existence of the target path when the event monitor is activated. At that time, if the target path does not exist, an error (SQLSTATE 428A3) is raised.

If an absolute path (a path that starts with the root directory on AIX, or a disk identifier on Windows) is specified, the specified path will be the one used. In environments other than DB2 pureScale, if a relative path (a path that does not start with the root) is specified, then the path relative to the DB2EVENT directory in the database directory will be used. In a DB2 pureScale environment, if a relative path is specified, then the path relative to the database owning directory in the database directory will be used.

It is possible to specify two or more event monitors that have the same target path. However, once one of the event monitors has been activated for the first time, and as long as the target directory is not empty, it will be impossible to activate any of the other event monitors.

file-options
Specifies the options for the file format.

MAXFILES NONE
Specifies that there is no limit to the number of event files that the event monitor will create. This is the default.

MAXFILES *number-of-files*
Specifies that there is a limit on the number of event monitor files that will exist for a particular event monitor at any time. Whenever an event monitor has to create another file, it will check to make sure that the number of .evt files in the directory is less than *number-of-files*. If this limit has already been reached, then the event monitor will turn itself off.

If an application removes the event files from the directory after they have been written, then the total number of files that an event monitor can produce can exceed *number-of-files*. This option has

been provided to allow a user to guarantee that the event data will not consume more than a specified amount of disk space.

MAXFILESIZE *pages*
Specifies that there is a limit to the size of each event monitor file. Whenever an event monitor writes a new event record to a file, it checks that the file will not grow to be greater than *pages* (in units of 4K pages). If the resulting file would be too large, then the event monitor switches to the next file. The default for this option is:

- Windows - 200 4K pages
- UNIX - 1000 4K pages

The number of pages must be greater than at least the size of the event buffer in pages. If this requirement is not met, then an error (SQLSTATE 428A4) is raised.

MAXFILESIZE NONE
Specifies that there is no set limit on a file's size. If MAXFILESIZE NONE is specified, then MAXFILES 1 must also be specified. This option means that one file will contain all of the event data for a particular event monitor. In this case the only event file will be 00000000.evt.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K pages). All event monitor file I/O is buffered to improve the performance of the event monitors. The larger the buffers, the less I/O will be performed by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event monitors. When the monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O.

The default size of each buffer is 4 pages (two 16K buffers are allocated). The minimum size is 1 page. The maximum size of the buffers is limited by the value of the MAXFILESIZE parameter, as well as the size of the monitor heap, because the buffers are allocated from that heap. If many event monitors are being used at the same time, increase the size of the **mon_heap_sz** database manager configuration parameter.

Event monitors that write their data to a pipe also have two internal (non-configurable) buffers that are each 1 page in size. These buffers are also allocated from the monitor heap (MON_HEAP). For each active event monitor that has a pipe target, increase the size of the database heap by 2 pages.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

NONBLOCKED
Specifies that each agent that generates an event should not wait for the event buffer to be written out to disk if the agent determines that both event buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of

BLOCKED event monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems.

APPEND
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will append the new event data to the existing stream of data files. When the event monitor is reactivated, it will resume writing to the event files as if it had never been turned off. APPEND is the default option.

The APPEND option does not apply at CREATE EVENT MONITOR time, if there is existing event data in the directory where the newly created event monitor is to write its event data.

REPLACE
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will erase all of the event files and start writing data to file 00000000.evt.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor has been activated, it can be deactivated only by using the SET EVENT MONITOR STATE statement or by stopping the instance.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the database partition on which the event monitor runs is activated. This is the default behavior of the threshold violations event monitor.

ON MEMBER *member-number*
Specifies the member on which a file or pipe event monitor is to run. When the monitoring scope is defined as LOCAL, data is collected only on the member. The I/O component will physically run on the specified member, writing records to the specified file or pipe. When the DB2 pureScale is enabled, -1 is the default. If the value is -1, it allows the I/O component to run from any active member. Additionally, in the event that the I/O component is no longer able to run on a given member, the event monitor will be restarted with the I/O component running on another available active member.

This clause is not valid for table event monitors. In a partitioned database environment, write-to-table event monitors will run and write events on all database partitions where table spaces for target tables are defined. In a DB2 pureScale environment, write-to-table event monitors will record events on all active members.

If this clause is not specified and DB2 pureScale is not enabled, the currently connected member is used. If this clause is not specified and DB2 pureScale is enabled, the I/O component is able to run on any currently connected member.

LOCAL
The event monitor reports only on the member that is running. It gives a partial trace of the database activity. This is the default.

This clause is valid for file or pipe monitors. It is not valid for table event monitors.

GLOBAL is not a valid scope for this type of event monitor.

Rules

- The THRESHOLD VIOLATIONS event type cannot be combined with any other event types in a particular event monitor definition.

Notes

- Event monitor definitions are recorded in the SYSCAT.EVENTMONITORS catalog view. The events themselves are recorded in the SYSCAT.EVENTS catalog view. The names of target tables are recorded in the SYSCAT.EVENTTABLES catalog view.
- If the member on which the event monitor is to run is not active, event monitor activation occurs when that member next activates.
- After an event monitor is activated, it behaves like an autostart event monitor until that event monitor is explicitly deactivated or the instance is recycled. That is, if an event monitor is active when a member is deactivated, and that member is subsequently reactivated, the event monitor is also explicitly reactivated.
- ***Write to table event monitors***: General notes:
  - All target tables are created when the CREATE EVENT MONITOR statement executes.
  - If the creation of a table fails for any reason, an error is passed back to the application program, and the CREATE EVENT MONITOR statement fails.
  - A target table can only be used by one event monitor. During CREATE EVENT MONITOR processing, if a target table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view.
  - During CREATE EVENT MONITOR processing, if a table already exists, but is *not* defined for use by another event monitor, no table is created, and processing continues. A warning is passed back to the application program.
  - Any table spaces must exist before the CREATE EVENT MONITOR statement is executed. The CREATE EVENT MONITOR statement does not create table spaces.
  - If specified, the LOCAL and GLOBAL keywords are ignored. With WRITE TO TABLE event monitors, an event monitor output process or thread is started on each member in the instance, and each of these processes reports data only for the member on which it is running.
  - The following event types from the flat monitor log file or pipe format are not recorded by write to table event monitors:
    - LOG_STREAM_HEADER
    - LOG_HEADER
    - DB_HEADER (Elements db_name and db_path are not recorded. The element conn_time is recorded in CONTROL.)
  - In a partitioned database environment, data is only written to target tables on the database partitions where their table spaces exist. If a table space for a target table does not exist on some database partition, data for that target table is ignored. This behavior allows users to choose a subset of database partitions for monitoring, by creating a table space that exists only on certain database partitions. In a DB2 pureScale environment, data will be written from every member.

In a partitioned database environment, if some target tables do not reside on a database partition, but other target tables do reside on that same database partition, only the data for the target tables that do reside on that database partition is recorded.

– Users must manually prune all target tables.

Table Columns:

– Column names in a table match an event monitor element identifier. Any event monitor element that does not have a corresponding target table column is ignored.

– Use the db2evtbl command to build a CREATE EVENT MONITOR command that includes a complete list of elements for a group.

– The types of columns being used for monitor elements correlate to the following mapping:

```
SQLM_TYPE_STRING                     CHAR[n], VARCHAR[n] or CLOB(n)
                                     (If the data in the event monitor
                                     record exceeds n bytes,
                                     it is truncated.)
SQLM_TYPE_U8BIT and SQLM_TYPE_8BIT   SMALLINT, INTEGER or BIGINT
SQLM_TYPE_16BIT and SQLM_TYPE_U16BIT SMALLINT, INTEGER or BIGINT
SQLM_TYPE_32BIT and SQLM_TYPE_U32BIT INTEGER or BIGINT
SQLM_TYPE_U64BIT and SQLM_TYPE_64BIT BIGINT
sqlm_timestamp                       TIMESTAMP
sqlm_time(elapsed time)              BIGINT
sqlca:
  sqlerrmc                           VARCHAR[72]
  sqlstate                           CHAR[5]
  sqlwarn                            CHAR[11]
  other fields                       INTEGER or BIGINT
```

– Columns are defined to be NOT NULL.

– Unlike other target tables, the columns in the CONTROL table do not match monitor element identifiers. Columns are defined as follows:

```
Column Name      Data Type     Nullable  Description
-----------      ---------     --------  -----------
PARTITION_KEY    INTEGER       N         Distribution key (partitioned
                                         database only)
PARTITION_NUMBER INTEGER       N         Database partition number
                                         (partitioned database only)
EVMONNAME        VARCHAR(128)  N         Name of the event monitor
MESSAGE          VARCHAR(128)  N         Describes the nature of
                                         the MESSAGE_TIME column.
                                         This can be one of
                                         the following values:
                                         - FIRST_CONNECT (the time
                                           of the first connect to the
                                           database after activation)
                                         - EVMON_START (the time that
                                           the event monitor listed
                                           in EVMONNAME was started)
                                         - OVERFLOWS:n (denotes that
                                           n records were discarded
                                           because of buffer overflow)
                                         - LAST_DROPPED_RECORD
                                           (the last time that
                                           an overflow occurred)
MESSAGE_TIME   TIMESTAMP       N         Timestamp
```

– In a partitioned database environment, the first column of each table is named PARTITION_KEY, is NOT NULL, and is of type INTEGER. This column is used as the distribution key for the table. The value of this column is chosen so that each event monitor process inserts data into the database partition on which the process is running; that is, insert operations are

performed locally on the database partition where the event monitor process is running. On any database partition, the PARTITION_KEY field will contain the same value. This means that if a database partition is dropped and data redistribution is performed, all data on the dropped database partition will go to one other database partition instead of being evenly distributed. Therefore, before removing a database partition, consider deleting all table rows on that database partition.

– In a partitioned database environment, a column named PARTITION_NUMBER can be defined for each table. This column is NOT NULL and is of type INTEGER. It contains the number of the database partition on which the data was inserted. Unlike the PARTITION_KEY column, the PARTITION_NUMBER column is not mandatory. The PARTITION_NUMBER column is not allowed in a non-partitioned database environment.

Table Attributes:

– Default table attributes are used. Besides distribution key (partitioned databases only), no extra options are specified when creating tables.
– Indexes on the table can be created.
– Extra table attributes (such as volatile, RI, triggers, constraints, and so on) can be added, but the event monitor process (or thread) will ignore them.
– If "not logged initially" is added as a table attribute, it is turned off at the first COMMIT, and is not set back on.

Event Monitor Activation:

– When an event monitor activates, all target table names are retrieved from the SYSCAT.EVENTTABLES catalog view.
– In a partitioned database environment, activation processing occurs on every database partition of the instance. On a particular database partition, activation processing determines the table spaces and database partition groups for each target table. The event monitor only activates on a database partition if at least one target table exists on that database partition. Moreover, if some target table is not found on a database partition, that target table is flagged so that data destined for that table is dropped during runtime processing.
– If a target table does not exist when the event monitor activates (or, in a partitioned database environment, if the table space does not reside on a database partition), activation continues, and data that would otherwise be inserted into this table is ignored.
– Activation processing validates each target table. If validation fails, activation of the event monitor fails, and messages are written to the administration log.
– During activation in a partitioned database environment, the CONTROL table rows for FIRST_CONNECT and EVMON_START are only inserted on the catalog database partition. This requires that the table space for the control table exist on the catalog database partition. If it does not exist on the catalog database partition, these inserts are not performed.
– In a partitioned database environment, if a partition is not yet active when a write to table event monitor is activated, the event monitor will be activated the next time that partition is activated.

Run Time:

– An event monitor runs with DATAACCESS authority.
– If, while an event monitor is active, an insert operation into a target table fails:

    - Uncommitted changes are rolled back.
    - A message is written to the administration log.
    - The event monitor is deactivated.
  – If an event monitor is active, it performs a local COMMIT when it has finished processing an event monitor buffer.
  – In an environment other than a partitioned database or a DB2 pureScale environment, all write to table event monitors are deactivated when the last application terminates (and the database has not been explicitly activated). In a DB2 pureScale environment, write to table event monitors are deactivated on a given member when the member stops and is reactivated when the member restarts. In a partitioned database environment, write to table event monitors are deactivated when the catalog partition deactivates.
  – The DROP EVENT MONITOR statement does not drop target tables.
  – Whenever a write-to-table event monitor activates, it will acquire IN table locks on each target table in order to prevent them from being modified while the event monitor is active. Table locks are maintained on all tables while the event monitor is active. If exclusive access is required on any of the target tables (for example, when a utility is to be run), first deactivate the event monitor to release the table locks before attempting such access.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  – DBPARTITIONNUM or NODE can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  – Commas can be used to separate multiple options in the *target-table-options* clause

Example

Define a threshold violation event monitor named DB2THRESHOLDVIOLATIONS

```
CREATE EVENT MONITOR DB2THRESHOLDVIOLATIONS
    FOR THRESHOLD VIOLATIONS
    WRITE TO TABLE
    THRESHOLDVIOLATIONS (TABLE THRESHOLDVIOLATIONS_DB2THRESHOLDVIOLATIONS
                         IN USERSPACE1
                         PCTDEACTIVATE 100),
    CONTROL (TABLE CONTROL_DB2THRESHOLDVIOLATIONS
             IN USERSPACE1
             PCTDEACTIVATE 100)
    AUTOSTART;
```

CREATE EVENT MONITOR (unit of work)

The CREATE EVENT MONITOR (unit of work) statement creates an event monitor that will record events when a unit of work completes.

Invocation

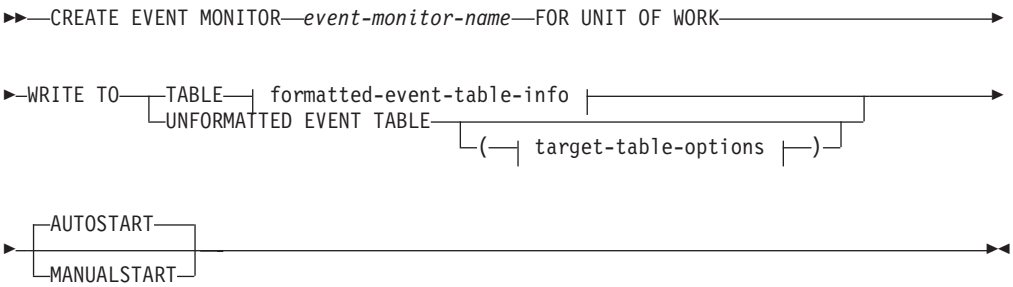
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

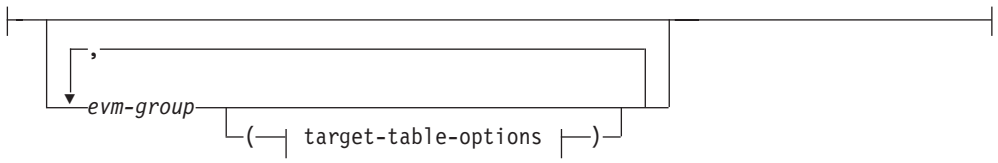
The privileges held by the authorization ID of the statement must include one of the following authorities:

- DBADM authority
- SQLADM authority

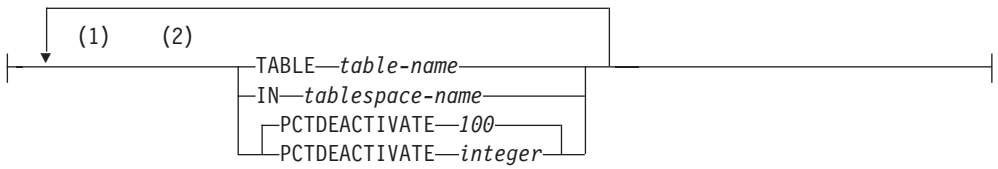
Syntax



formatted-event-table-info:



target-table-options:



Notes:

1 Each table option can be specified a maximum of one time (SQLSTATE 42613).

2 Clauses can be separated with a space or a comma.

Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

UNIT OF WORK
Specifies that this passive event monitor will record an event whenever a unit of work is completed (that is, whenever there is a commit or rollback).

The creation of the unit of work event monitor does not indicate that the unit of work data will be collected immediately. The actual unit of work event of interest is controlled at the workload level.

WRITE TO
Specifies the target for the data.

TABLE
Indicates that the target for the event monitor data is a set of database tables. The event monitor separates the data stream into one or more logical data groups and inserts each group into a separate table. Data for groups having a target table is kept, whereas data for groups not having a target table is discarded. Each monitor element contained within a group is mapped to a table column with the same name. Only elements that have a corresponding table column are inserted into the table. Other elements are discarded.

formatted-event-table-info
Defines the target formatted event tables for the event monitor. This clause should specify each grouping that is to be recorded. However, if no *evm-group* clauses are specified, all groups for the event monitor type are recorded.

evm-group
Identifies a logical data group for which a target table is being defined. The value depends upon the type of event monitor, as shown in the following table:

Type of Event Monitor	evm-group Value
Unit of work	• UOW • UOW_METRICS • UOW_PACKGE_LIST • UOW_EXECUTABLE_LIST • CONTROL

UNFORMATTED EVENT TABLE
Specifies that the target for the event monitor is an unformatted event table. The unformatted event table is used to store collected unit of work event monitor data. Data is stored in its original binary format within an inlined BLOB column. The BLOB column can contain multiple binary records of different types. The data in the BLOB column is not in a readable format and requires conversion, through use of the **db2evmonfmt** Java-based tool, EVMON_FORMAT_UE_TO_XML table function, or EVMON_FORMAT_UE_TO_TABLES procedure, into a consumable format such as an XML document or a relational table.

target-table-options
Identifies options for the target table. If a value for **target-table-options** is not specified, CREATE EVENT MONITOR processing proceeds as follows:

- A derived table name is used (as explained in the description for TABLE *table-name*).
- A default table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.
- PCTDEACTIVATE is set to 100.

TABLE *table-name*
Specifies the name of the target table. The target table must be a non-partitioned table. If the name is unqualified, the table schema defaults to the value in the CURRENT SCHEMA special register. For an unformatted event table if a name is not provided, the unqualified name is equal to the *event-monitor-name*, that is, the unformatted event table will be named after the event monitor. For a formatted event table if no name is provided, the unqualified name is derived from *evm-group* and *event-monitor-name* as follows:

```
substring(evm-group CONCAT '_'
  CONCAT event-monitor-name,1,128)
```

IN *tablespace-name*
Specifies the table space in which the table is to be created. The CREATE EVENT MONITOR FOR UNIT OR WORK statement does not create table spaces.

If a table space name is not provided, the table space is chosen using the same process as when a table is created without a table space name using CREATE TABLE.

Since the page size affects the INLINE LOB lengths used, consider specifying a table space with as large a page size as possible in order to improve the INSERT performance of the event monitor.

PCTDEACTIVATE *integer*
If a table for the event monitor is being created in an automatic storage (non-temporary) or DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. The specified value, which represents a percentage, can range from 0 to 100, where 100 means that the event monitor deactivates when the table space becomes completely full. The default value assumed is 100 if PCTDEACTIVATE is not specified. This option is ignored for SMS table spaces.

Important: If the target table space has auto-resize enabled, set PCTDEACTIVATE parameter to 100. Alternatively, omit this clause entirely to have the default of 100 apply. Otherwise, the event monitor might deactivate unexpectedly if the table space reaches the threshold specified by PCTDEACTIVTATE before the table space is automatically resized.

AUTOSTART
Specifies that the event monitor is to be automatically activated whenever the database partition on which the event monitor runs is activated. This is the default behavior of the unit of work event monitor.

MANUALSTART
Specifies that the event monitor must be activated manually using the SET EVENT MONITOR STATE statement. After a MANUALSTART event monitor

has been activated, it can be deactivated only by using the SET EVENT MONITOR STATE statement or by stopping the instance.

Notes

- The table is created when the CREATE EVENT MONITOR FOR UNIT OF WORK statement executes, if it doesn't already exist.
- During CREATE EVENT MONITOR FOR UNIT OF WORK processing, if a table is found to have already been defined for use by another event monitor, the CREATE EVENT MONITOR FOR UNIT OF WORK statement fails, and an error is passed back to the application program. A table is defined for use by another event monitor if the table name matches a value found in the SYSCAT.EVENTTABLES catalog view. If the table exists and is not defined for use by another event monitor, then no table is created, any other table **target-table-options** parameters are ignored, and processing continues. A warning is passed back to the application program.
- Dropping the event monitor will not drop any tables. Any associated tables must be manually dropped after the event monitor is dropped.
- Lock event data is not automatically pruned from either unformatted event tables or regular tables created by this event monitor. An option for pruning data from UE tables is available when using the EVMON_FORMAT_UE_TO_TABLES procedure. For event monitors that write to regular tables, event data must be pruned manually.
- For unformatted event tables event data is inserted into the table into an inlined BLOB data column. Normally, BLOB data is stored in a separate LOB table space and can experience additional performance overhead as a result. When inlined into the data page of the base table, the BLOB data does not experience this overhead. The DB2 database manager will automatically inline the BLOB data portion of an unformatted event table record if the size of the BLOB data is less than the table space page size minus the record prefix. Therefore to achieve high efficiency and application throughput, it is suggested that you create the event monitor in as large a table space as possible up to and including a 32 KB table space and associated bufferpool.
- Create only one unit of work event monitor per database and not create multiple unit of work event monitors on the same database.
- In a partitioned database environment, data is written only to target tables on the database partitions where their table spaces exist. If a table space for a target unformatted event table does not exist on some database partition, data for that target table is ignored. This behavior allows users to choose a subset of database partitions for monitoring to be chosen, by creating a table space that exists only on certain database partitions.
- In a multi-member environment, data is only written to target tables on the member where work occurs within the unit of work.
- In a partitioned database environment, if some target tables do not reside on a database partition, but other target tables do reside on that same database partition, only the data for the target tables that do reside on that database partition is recorded.
- The unit of work event monitor is not affected by the unit or work event monitor switch. The unit of work event monitor switch is not changed when a unit or work event monitor is created, and the contents of the unit or work event monitor are not affected by changes to the unit of work event monitor switch.
- The FLUSH EVENT MONITOR statement is not applicable to this event monitor and will have no effect when issued against it.

- Creation of the unit of work event monitor does not cause events to be written to the event monitor. The unit of work event monitor must be activated with SET EVENT MONITOR STATE, and the unit of work data must be collected by either altering the appropriate workload to specify COLLECT UNIT OF WORK DATA or setting the **mon_uow_data** database configuration parameter to a value other than NONE.
- When using unformatted event tables, create the unit of work event monitor in a table space with at least 8 KB page size to ensure that the event data is contained within the inlined BLOB column of the unformatted event table. If the BLOB column is not inlined, then the performance of writing and reading the events to the unformatted event table might not be efficient.

Examples

- *Example 1:* This example creates a unit of work event monitor UOWEVMON that collects data for unit of work events that occur on the database of creation, and writes data tables using default table names:

```
CREATE EVENT MONITOR UOWEVMON
  FOR UNIT OF WORK
  WRITE TO TABLE
```

  This event monitor writes its output to the following tables:

  UOW_UOWEVMON

  UOW_METRICS_UOWEVMON

  UOW_PACKAGE_LIST_UOWEVMON

  UOW_EXECUTABLE_LIST_UOWEVMON

  UOW_CONTROL_UOWEVMON

  Note: Whether the tables for package list and executable list information are populated with data is dependent on whether you specify that that data is to be collected. You control the collection of this data is using the mon_uow_pkglist or mon_uow_execlist configuration parameters, or with the appropriate COLLECT UNIT OF WORK DATA clause on the CREATE or ALTER WORKLOAD statements.

- *Example 2:* This example creates a unit of work event monitor UOWEVMON that will collect unit of work events that occur on the database of creation and store it in the unformatted event table GREG.UOWEVENTS.

```
CREATE EVENT MONITOR UOWEVMON
  FOR UNIT OF WORK
  WRITE TO UNFORMATTED EVENT TABLE (TABLE GREG.UOWEVENTS)
```

- *Example 3:* This example creates a unit of work event monitor UOWEVMON that will collect unit of work events that occur on the database of creation and store it in the unformatted event table GREG.UOWEVENTS in table space APPSPACE. The event monitor will deactivate when the table space becomes 85% full.

```
CREATE EVENT MONITOR UOWEVMON
  FOR UNIT OF WORK
  WRITE TO UNFORMATTED EVENT TABLE
     (TABLE GREG.UOWEVENTS IN APPSPACE PCTDEACTIVATE 85)
```

CREATE FUNCTION

The CREATE FUNCTION statement is used to register or define a user-defined function or function template at the current server.

There are five different types of functions that can be created using this statement. Each of these is described separately.

- External Scalar. The function is written in a programming language and returns a scalar value. The external executable is registered in the database, along with various attributes of the function.
- External Table. The function is written in a programming language and returns a complete table. The external executable is registered in the database along with various attributes of the function.
- OLE DB External Table. A user-defined OLE DB external table function is registered in the database to access data from an OLE DB provider.
- Sourced or Template. A source function is implemented by invoking another function (either built-in, external, SQL, or source) that is already registered in the database.

  It is possible to create a partial function, called a *function template*, which defines what types of values are to be returned, but which contains no executable code. The user maps it to a data source function within a federated system, so that the data source function can be invoked from a federated database. A function template can be registered only with an application server that is designated as a federated server.
- SQL Scalar, Table or Row. The function body is written in SQL and defined together with the registration in the database. It returns a scalar value, a table, or a single row.

The CREATE FUNCTION statement can be submitted in obfuscated form. In an obfuscated statement, only the function name and its parameters are readable. The rest of the statement is encoded in such a way that is not readable but can be decoded by the database server. Obfuscated statements can be produced by calling the DBMS_DDL.WRAP function.

CREATE FUNCTION (external scalar)

The CREATE FUNCTION (External Scalar) statement is used to register a user-defined external scalar function at the current server. A *scalar function* returns a single value each time it is invoked, and is in general valid wherever an SQL expression is valid.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CREATE_EXTERNAL_ROUTINE authority on the database and at least one of the following authorities:
  - IMPLICIT_SCHEMA authority on the database, if the schema name of the function does not refer to an existing schema
  - CREATEIN privilege on the schema, if the schema name of the function refers to an existing schema
- DBADM authority

Group privileges are not considered for any table or view specified in the CREATE FUNCTION statement.

To create a not-fenced function, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

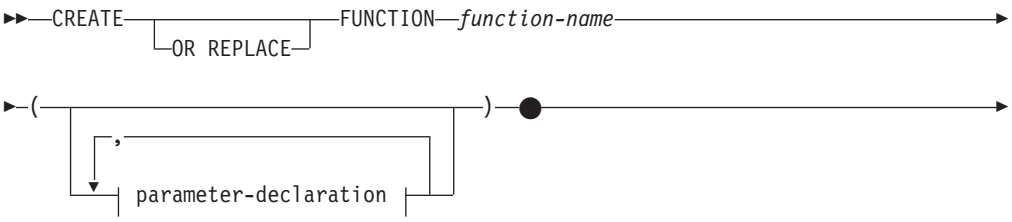
- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

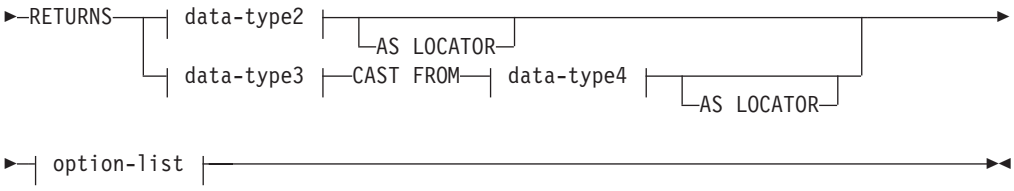
To create a fenced function, no additional authorities or privileges are required.

To replace an existing function, the authorization ID of the statement must be the owner of the existing function (SQLSTATE 42501).

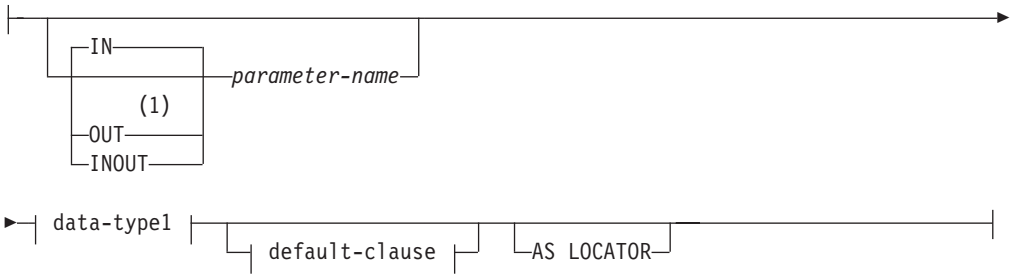
If the SECURED option is specified, the authorization ID of the statement must include SECADM or CREATE_SECURE_OBJECT authority (SQLSTATE 42501).

Syntax



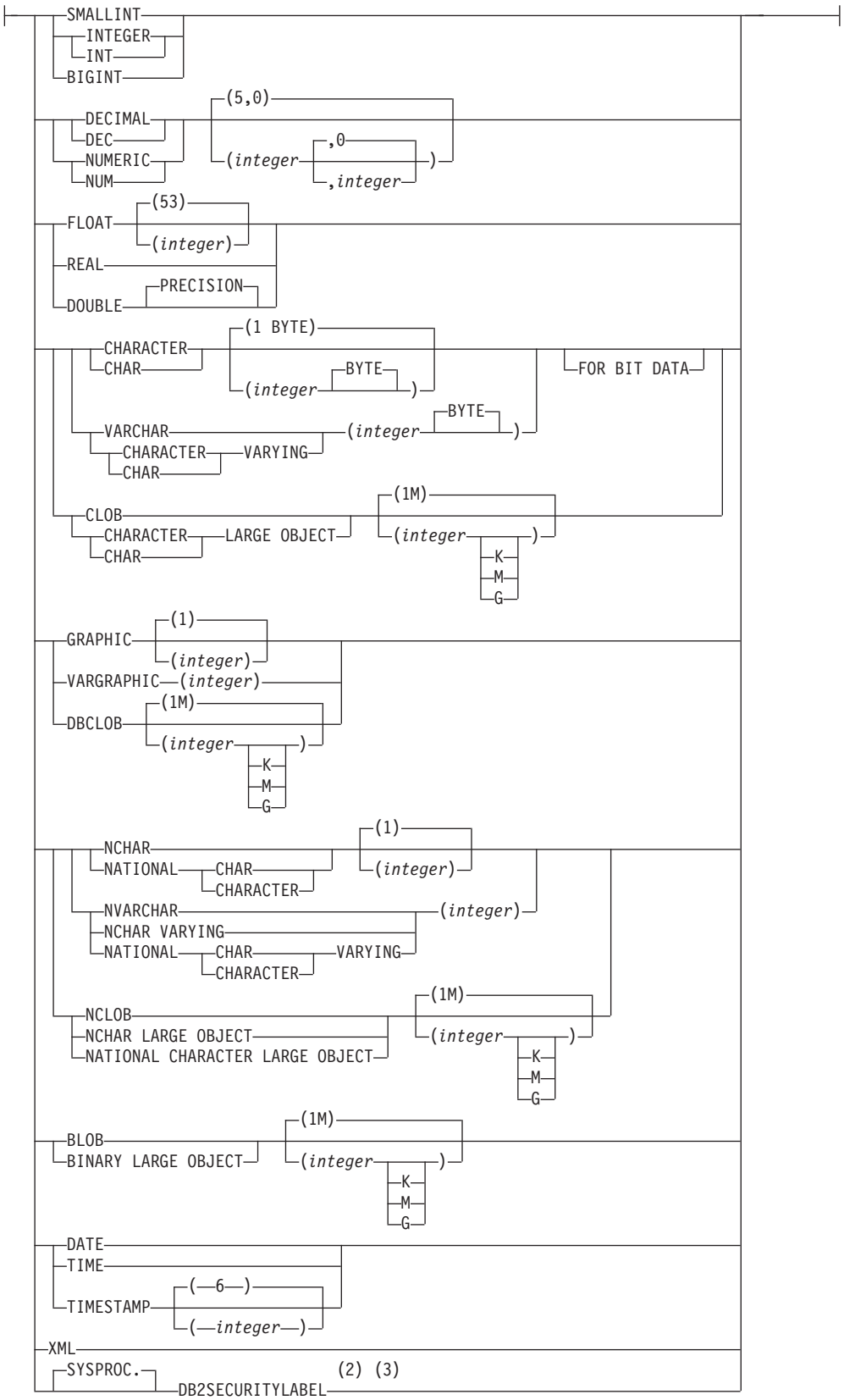


parameter-declaration:



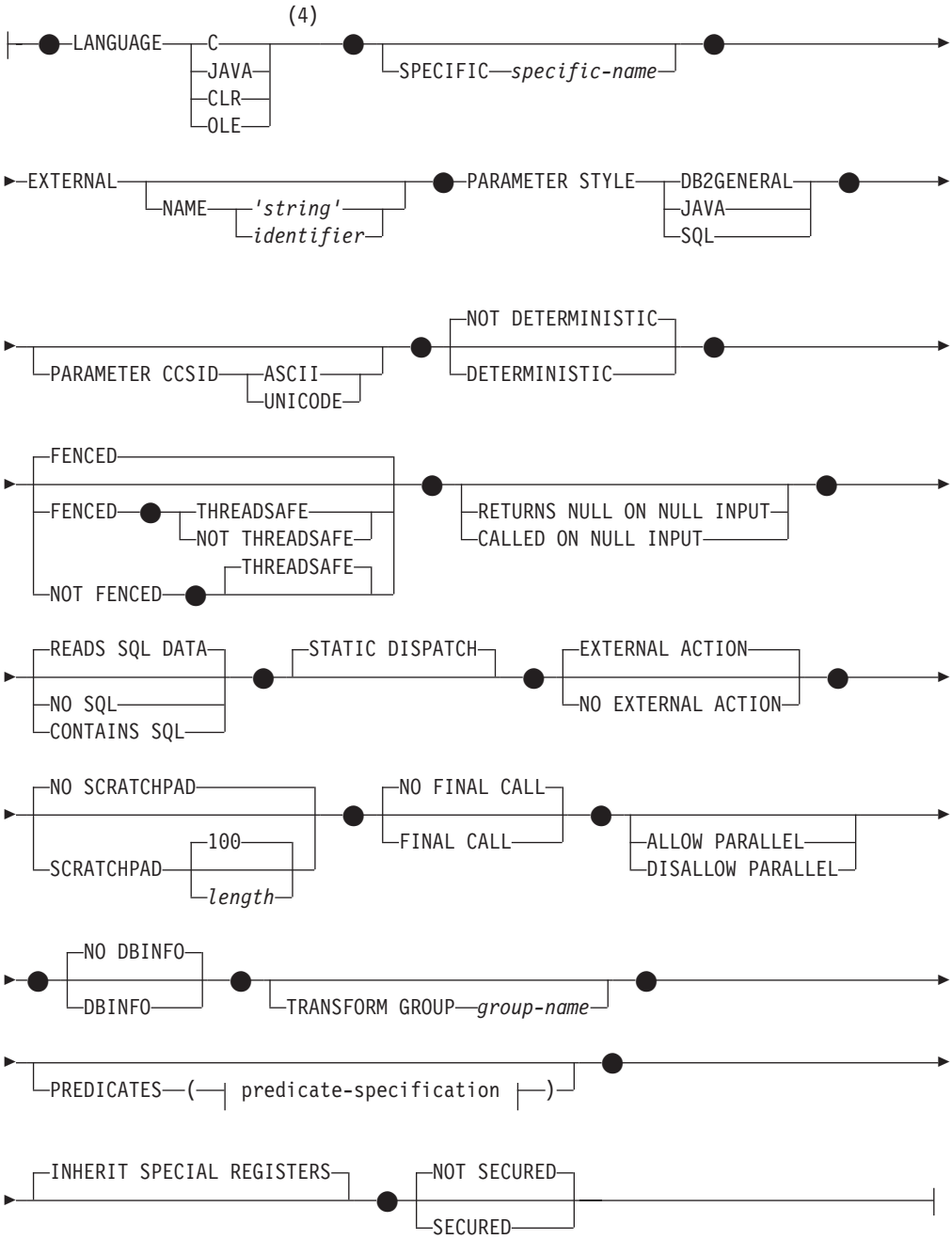
data-type1, data-type2, data-type3, data-type4:

built-in-type:

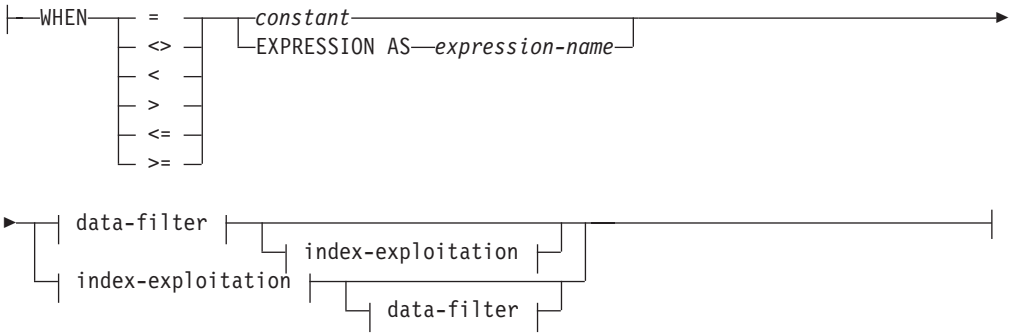

SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer)
(integer,0)
(integer,integer)
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
CHARACTER
CHAR
(1 BYTE)
(integer)
(integer BYTE)
FOR BIT DATA
VARCHAR
CHARACTER VARYING
CHAR VARYING
CLOB
CHARACTER LARGE OBJECT
CHAR LARGE OBJECT
(1M)
(integer)
(integer K)
(integer M)
(integer G)
GRAPHIC
(1)
(integer)
VARGRAPHIC(integer)
DBCLOB
(1M)
(integer)
(integer K)
(integer M)
(integer G)
NCHAR
NATIONAL CHAR
NATIONAL CHARACTER
(1)
(integer)
NVARCHAR
NCHAR VARYING
NATIONAL CHAR VARYING
NATIONAL CHARACTER VARYING
(integer)
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
(1M)
(integer)
(integer K)
(integer M)
(integer G)
BLOB
BINARY LARGE OBJECT
(1M)
(integer)
(integer K)
(integer M)
(integer G)
DATE
TIME
TIMESTAMP
(6)
(integer)
XML
SYSPROC.
DB2SECURITYLABEL
(2) (3)

default-clause:

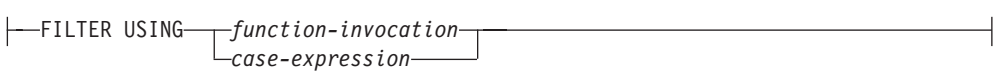
option-list:



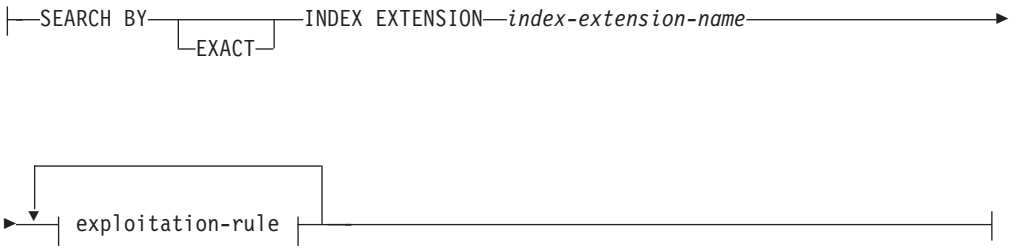
predicate-specification:



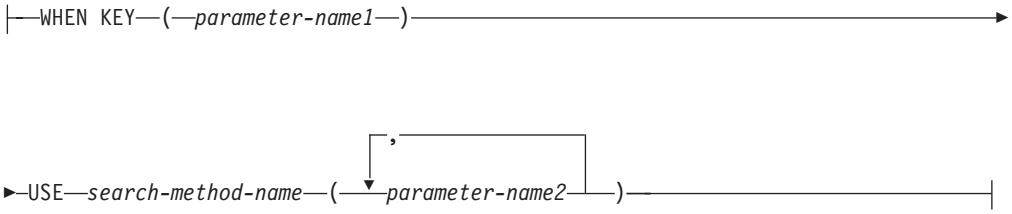
data-filter:



index-exploitation:



exploitation-rule:



Notes:

1 OUT and INOUT are valid only if the function has LANGUAGE C.

2 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

3 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.

4 LANGUAGE SQL is also supported.

Description

OR REPLACE

Specifies to replace the definition for the function if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the function are not affected. This option can be specified only by the owner of the object. This option is ignored if a definition for the function does not exist at the current server. To replace an existing function, the specific name and function name of the new definition must be the same as the specific name and function name of the old definition, or the signature of the new definition must match the signature of the old definition. Otherwise, a new function is created.

If the function is referenced in the definition of a row permission or a column mask, the function cannot be replaced (SQLSTATE 42893).

function-name

Names the function being defined. It is a qualified or unqualified name that designates a function. The unqualified form of *function-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier. The qualified name must not be the same as the data type of the first parameter, if that first parameter is a structured type.

The name, including the implicit or explicit qualifiers, together with the number of parameters and the data type of each parameter (without regard for any length, precision or scale attributes of the data type) must not identify a function or method described in the catalog (SQLSTATE 42723). The unqualified name, together with the number and data types of the parameters, while of course unique within its schema, need not be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS';. Otherwise, an error (SQLSTATE 42939) is raised.

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *function-name*. The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators. Failure to observe this rule will lead to an error (SQLSTATE 42939).

In general, the same name can be used for more than one function if there is some difference in the signature of the functions.

Although there is no prohibition against it, an external user-defined function should not be given the same name as a built-in function, unless it is an intentional override. To give a function having a different meaning the same name (for example, LENGTH, VALUE, MAX), with consistent arguments, as a built-in scalar or aggregate function, is to invite trouble for dynamic SQL statements, or when static SQL applications are rebound; the application may fail, or perhaps worse, may appear to run successfully while providing a different result.

(*parameter-declaration***,...)**

Identifies the number of input parameters of the function, and specifies the mode, name, data type, and optional default value of each parameter. One entry in the list must be specified for each parameter that the function expects to receive. Up to 90 parameters can be specified (SQLSTATE 54023).

You can register a function that has no parameters; the parentheses must still be coded, with no intervening data types. For example:

```
CREATE FUNCTION WOOFER() ...
```

No two identically-named functions within a schema are permitted to have exactly the same type for all corresponding parameters. Lengths, precisions, and scales are not considered in this type comparison. Therefore, CHAR(8) and CHAR(35) are considered to be the same type, as are DECIMAL(11,2) and DECIMAL (4,3). A weakly typed distinct type specified for a parameter is considered to be the same data type as the source type of the distinct type. For a Unicode database, CHAR(13) and GRAPHIC(8) are considered to be the same type. There is some further bundling of types that causes them to be treated as the same type for this purpose, such as DECIMAL and NUMERIC. A duplicate signature returns an error (SQLSTATE 42723).

IN | OUT | INOUT
Specifies the mode of the parameter. If an error is returned by the function, OUT parameters are undefined and INOUT parameters are unchanged. The default is IN.

IN Identifies the parameter as an input parameter to the function. Any changes made to the parameter within the function are not available to the invoking context when control is returned.

OUT
Identifies the parameter as an output parameter for the function.

The function must be defined with LANGUAGE C (SQLSTATE 42613).

The function can be referenced only on the right side of an assignment statement that is in a compound SQL (compiled) statement, and the function reference cannot be part of an expression (SQLSTATE 42887).

INOUT
Identifies the parameter as both an input and output parameter for the function.

The function must be defined with LANGUAGE C (SQLSTATE 42613).

The function can be referenced only on the right side of an assignment statement that is in a compound SQL (compiled) statement, and the function reference cannot be part of an expression (SQLSTATE 42887).

parameter-name
Specifies an optional name for the parameter. Parameter names are required to reference the parameters of a function in the *index-exploitation* clause of a predicate specification. The name cannot be the same as any other *parameter-name* in the parameter list (SQLSTATE 42734).

data-type1
Specifies the data type of the parameter. The data type can be a built-in data type, a distinct type, a structured type, or a reference type. For a more complete description of each built-in data type, see "CREATE TABLE". Some data types are not supported in all languages. For details on the mapping between SQL data types and host language data types, see "Data types that map to SQL data types in embedded SQL applications".

- A datetime type parameter is passed as a character data type, and the data is passed in the ISO format.
- DECIMAL (and NUMERIC) are invalid with LANGUAGE C and OLE (SQLSTATE 42815).

- DECFLOAT is invalid with LANGUAGE C, COBOL, CLR, JAVA, and OLE (SQLSTATE 42815).
- XML is invalid with LANGUAGE OLE.
- Because the XML value that is seen inside a function is a serialized version of the XML value that is passed as a parameter in the function call, parameters of type XML must be declared using the syntax `XML AS CLOB(n)`.
- CLR does not support DECIMAL scale greater than 28 (SQLSTATE 42613).
- Array types cannot be specified (SQLSTATE 42815).

For a user-defined distinct type, the length, precision, or scale attributes for the parameter are those of the source type of the distinct type (those specified on CREATE TYPE). A distinct type parameter is passed as the source type of the distinct type. If the name of the distinct type is unqualified, the database manager resolves the schema name by searching the schemas in the SQL path.

For a user-defined structured type, the appropriate transform functions must exist in the associated transform group.

For a reference type, the parameter can be specified as REF(*type-name*) if the parameter is unscoped.

DEFAULT

Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression, or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see *default-clause* in the CREATE TABLE statement). Other special registers can be specified as the default by using an expression.

The expression can be any expression of the type described in "Expressions". If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified in the following situations:

- For INOUT or OUT parameters (SQLSTATE 42601)
- For a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB)
- For a parameter to a function definition that also specified a PREDICATES clause (SQLSTATE 42613)

AS LOCATOR

Specifies that a locator to the value of the parameter is passed to the function instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB data type (SQLSTATE 42601). Passing locators instead of values can result in fewer bytes being passed to the function, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted, nor does it affect the function signature, which is used in function resolution.

If the function is FENCED and has the NO SQL option, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

RETURNS
This mandatory clause identifies the output of the function.

data-type2
Specifies the data type of the output.

In this case, exactly the same considerations apply as for the parameters of external functions described previously in *data-type1* for function parameters.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be passed from the UDF instead of the actual value.

data-type3 **CAST FROM** *data-type4*
Specifies the data type of the output.

This form of the RETURNS clause is used to return a different data type to the invoking statement from the data type that was returned by the function code. For example, in

```
CREATE FUNCTION GET_HIRE_DATE(CHAR(6))
  RETURNS DATE CAST FROM CHAR(10)
  ...
```

the function code returns a CHAR(10) value to the database manager, which, in turn, converts it to a DATE and passes that value to the invoking statement. The *data-type4* must be castable to the *data-type3* parameter. If it is not castable, an error (SQLSTATE 42880) is raised.

Since the length, precision or scale for *data-type3* can be inferred from *data-type4*, it not necessary (but still permitted) to specify the length, precision, or scale for parameterized types specified for *data-type3*. Instead empty parentheses may be used (for example VARCHAR() may be used). FLOAT() cannot be used (SQLSTATE 42601) since parameter value indicates different data types (REAL or DOUBLE).

Distinct types, array types, and structured types are not valid as the type specified in *data-type4* (SQLSTATE 42815).

The cast operation is also subject to runtime checks that might result in conversion errors being raised.

AS LOCATOR
For *data-type4* specifications that are LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be passed back from the UDF instead of the actual value.

SPECIFIC *specific-name*
Provides a unique name for the instance of the function that is being defined. This specific name can be used when sourcing on this function, dropping the function, or commenting on the function. It can never be used to invoke the function. The unqualified form of *specific-name* is an SQL identifier. The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another function instance or method specification that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* may be the same as an existing *function-name*.

If no qualifier is specified, the qualifier that was used for *function-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *function-name* or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmmssxxx.

EXTERNAL

This clause indicates that the CREATE FUNCTION statement is being used to register a new function based on code written in an external programming language and adhering to the documented linkage conventions and interface.

If NAME clause is not specified "NAME *function-name*" is assumed.

NAME *'string'*

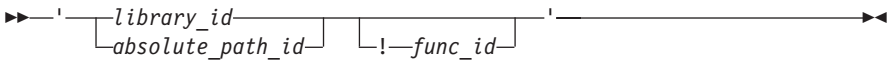
This clause identifies the name of the user-written code which implements the function being defined.

The 'string' option is a string constant with a maximum of 254 bytes. The format used for the string is dependent on the LANGUAGE specified.

- For LANGUAGE C:

  The *string* specified is the library name and function within the library, which the database manager invokes to execute the user-defined function being created. The library (and the function within the library) do not need to exist when the CREATE FUNCTION statement is executed. However, when the function is used in an SQL statement, the library and function within the library must exist and be accessible from the database server machine; otherwise, an error is returned (SQLSTATE 42724).

  The *string* can be specified as follows:



  Extraneous blanks are not permitted within the single quotation marks.

  library_id

  Identifies the library name containing the function. The database manager will look for the library as follows:

  – On UNIX systems, if 'myfunc' was given as the *library_id*, and the database manager is being run from /u/production, the database manager will look for the function in library /u/production/sqllib/function/myfunc.

  – On Windows operating systems, the database manager will look for the function in a directory path that is specified by the LIBPATH or PATH environment variable.

  absolute_path_id

  Identifies the full path name of the file containing the function.

  On UNIX systems, for example, '/u/jchui/mylib/myfunc' would cause the database manager to look in /u/jchui/mylib for the myfunc shared library.

  On Windows operating systems, 'd:\mylib\myfunc.dll' would cause the database manager to load the dynamic link library, myfunc.dll,

from the d:\mylib directory. If an absolute path ID is being used to identify the routine body, be sure to append the .dll extension.

! func_id
Identifies the entry point name of the function to be invoked. The ! serves as a delimiter between the library ID and the function ID.

On a UNIX system, for example, 'mymod!func8' would direct the database manager to look for the library $inst_home_dir/sqllib/function/mymod and to use entry point func8 within that library.

On Windows operating systems, 'mymod!func8' would direct the database manager to load the mymod.dll file and to call the func8() function in the dynamic link library (DLL).

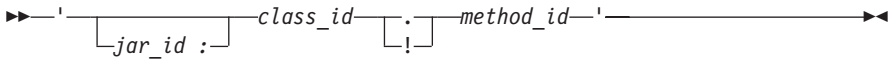
If the string is not properly formed, an error is returned (SQLSTATE 42878).

The body of every external function should be in a directory that is available on every database partition.

- For LANGUAGE JAVA:

  The *string* specified contains the optional jar file identifier, class identifier and method identifier, which the database manager invokes to execute the user-defined function being created. The class identifier and method identifier do not need to exist when the CREATE FUNCTION statement is executed. If a *jar_id* is specified, it must exist when the CREATE FUNCTION statement is executed. However, when the function is used in an SQL statement, the method identifier must exist and be accessible from the database server machine; otherwise, an error is returned (SQLSTATE 42724).

  The *string* can be specified as follows:



  Extraneous blanks are not permitted within the single quotation marks.

  jar_id
  Identifies the jar identifier given to the jar collection when it was installed in the database. It can be either a simple identifier, or a schema qualified identifier. Examples are 'myJar' and 'mySchema.myJar'.

  class_id
  Identifies the class identifier of the Java object. If the class is part of a package, the class identifier part must include the complete package prefix, for example, 'myPacks.UserFuncs'. The Java virtual machine will look in directory '.../myPacks/UserFuncs/' for the classes. On Windows operating systems, the Java virtual machine will look in directory '...\myPacks\UserFuncs\'.

  method_id
  Identifies the method name of the Java object to be invoked.

- For LANGUAGE CLR:

  The *string* specified represents the .NET assembly (library or executable), the class within that assembly, and the method within the class that the database manager invokes to execute the function being created. The module, class, and method do not need to exist when the CREATE FUNCTION statement is executed. However, when the function is used

in an SQL statement, the module, class, and method must exist and be accessible from the database server machine; otherwise, an error is returned (SQLSTATE 42724).

C++ routines that are compiled with the '/clr' compiler option to indicate that they include managed code extensions must be cataloged as 'LANGUAGE CLR' and not 'LANGUAGE C'. DB2 needs to know that the .NET infrastructure is being utilized in a user-defined function in order to make necessary runtime decisions. All user-defined functions using the .NET infrastructure must be cataloged as 'LANGUAGE CLR'.

The *string* can be specified as follows:

The name must be enclosed by single quotation marks. Extraneous blanks are not permitted.

assembly
Identifies the DLL or other assembly file in which the class resides. Any file extensions (such as .dll) must be specified. If the full path name is not given, the file must reside in the function directory of the DB2 install path (for example, c:\sqllib\function). If the file resides in a subdirectory of the install function directory, the subdirectory can be given before the file name rather than specifying the full path. For example, if your install directory is c:\sqllib and your assembly file is c:\sqllib\function\myprocs\mydotnet.dll, it is only necessary to specify 'myprocs\mydotnet.dll' for the assembly. The case sensitivity of this parameter is the same as the case sensitivity of the file system.

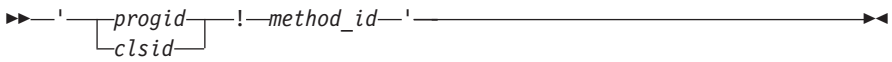
class_id
Specifies the name of the class within the given assembly in which the method that is to be invoked resides. If the class resides within a namespace, the full namespace must be given in addition to the class. For example, if the class EmployeeClass is in namespace MyCompany.ProcedureClasses, then MyCompany.ProcedureClasses.EmployeeClass must be specified for the class. Note that the compilers for some .NET languages will add the project name as a namespace for the class, and the behavior may differ depending on whether the command line compiler or the GUI compiler is used. This parameter is case sensitive.

method_id
Specifies the method within the given class that is to be invoked. This parameter is case sensitive.

- For LANGUAGE OLE:

  The *string* specified is the OLE programmatic identifier (progid) or class identifier (clsid), and method identifier, which the database manager invokes to execute the user-defined function being created. The programmatic identifier or class identifier, and method identifier do not need to exist when the CREATE FUNCTION statement is executed. However, when the function is used in an SQL statement, the method identifier must exist and be accessible from the database server machine; otherwise, an error is returned (SQLSTATE 42724).

  The *string* can be specified as follows:



Extraneous blanks are not permitted within the single quotation marks.

progid
Identifies the programmatic identifier of the OLE object.

progid is not interpreted by the database manager but only forwarded to the OLE APIs at run time. The specified OLE object must be creatable and support late binding (also called IDispatch-based binding).

clsid
Identifies the class identifier of the OLE object to create. It can be used as an alternative for specifying a *progid* in the case that an OLE object is not registered with a progid. The *clsid* has the form:

{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}

where 'n' is an alphanumeric character. *clsid* is not interpreted by the database manager but only forwarded to the OLE APIs at run time.

method_id
Identifies the method name of the OLE object to be invoked.

NAME *identifier*
This *identifier* specified is an SQL identifier. The SQL identifier is used as the *library-id* in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C.

LANGUAGE
This mandatory clause is used to specify the language interface convention to which the user-defined function body is written.

C This means the database manager will call the user-defined function as if it were a C function. The user-defined function must conform to the C language calling and linkage convention as defined by the standard ANSI C prototype.

JAVA This means the database manager will call the user-defined function as a method in a Java class.

CLR This means the database manager will call the user-defined function as a method in a .NET class. At this time, LANGUAGE CLR is only supported for user-defined functions running on Windows operating systems. NOT FENCED cannot be specified for a CLR routine (SQLSTATE 42601).

OLE This means the database manager will call the user-defined function as if it were a method exposed by an OLE automation object. The user-defined function must conform with the OLE automation data types and invocation mechanism, as described in the *OLE Automation Programmer's Reference*.

LANGUAGE OLE is only supported for user-defined functions stored in DB2 for Windows operating systems. THREADSAFE may not be specified for UDFs defined with LANGUAGE OLE (SQLSTATE 42613).

PARAMETER STYLE
This clause is used to specify the conventions used for passing parameters to and returning the value from functions.

DB2GENERAL
Used to specify the conventions for passing parameters to and returning the value from external functions that are defined as a method in a Java class. This can only specified when LANGUAGE JAVA is used.

The value DB2GENRL may be used as a synonym for DB2GENERAL.

JAVA
This means that the function will use a parameter passing convention that conforms to the Java language and SQLJ Routines specification. This can only be specified when LANGUAGE JAVA is used, no structured data types are specified as parameters, and no CLOB, BLOB, or DBCLOB data types are specified as return types (SQLSTATE 429B8). PARAMETER STYLE JAVA functions do not support the FINAL CALL, SCRATCHPAD, or DBINFO clause.

SQL
Used to specify the conventions for passing parameters to and returning the value from external functions that conform to C language calling and linkage conventions, methods exposed by OLE automation objects, or public static methods of a .NET object. This must be specified when LANGUAGE C, LANGUAGE CLR, or LANGUAGE OLE is used.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the function. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031). When the function is invoked, the application code page for the function is the database code page.

UNICODE
Specifies that string data is encoded in Unicode. If the database is a Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8. In either case, when the function is invoked, the application code page for the function is 1208.

If the database is not a Unicode database, and a function with PARAMETER CCSID UNICODE is created, the function cannot have any graphic types, the XML type, or user-defined types (SQLSTATE 560C1).

If the database is not a Unicode database, and the alternate collating sequence has been specified in the database configuration, functions can be created with either PARAMETER CCSID ASCII or PARAMETER CCSID UNICODE. All string data passed into and out of the function will be converted to the appropriate code page.

This clause cannot be specified with LANGUAGE OLE, LANGUAGE JAVA, or LANGUAGE CLR (SQLSTATE 42613).

DETERMINISTIC or NOT DETERMINISTIC
This optional clause specifies whether the function always returns the same

results for given argument values (DETERMINISTIC) or whether the function depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC function must always return the same result from successive invocations with identical inputs. Optimizations taking advantage of the fact that identical inputs always produce the same results are prevented by specifying NOT DETERMINISTIC. An example of a NOT DETERMINISTIC function would be a random-number generator. An example of a DETERMINISTIC function would be a function that determines the square root of the input.

FENCED or NOT FENCED

This clause specifies whether or not the function is considered "safe" to run in the database manager operating environment's process or address space.

If a function is registered as FENCED, the database manager protects its internal resources (for example, data buffers) from access by the function. Most functions will have the option of running as FENCED or NOT FENCED. In general, a function running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for functions not adequately coded, reviewed and tested can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED user-defined functions are used.

Only FENCED can be specified for a function with LANGUAGE OLE or NOT THREADSAFE (SQLSTATE 42613).

If the function is FENCED and has the NO SQL option, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

Either SYSADM authority, DBADM authority, or a special authority (CREATE_NOT_FENCED_ROUTINE) is required to register a user-defined function as NOT FENCED.

LANGUAGE CLR user-defined functions cannot be created when specifying the NOT FENCED clause (SQLSTATE 42601).

THREADSAFE or NOT THREADSAFE

Specifies whether the function is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the function is defined with LANGUAGE other than OLE:

- If the function is defined as THREADSAFE, the database manager can invoke the function in the same process as other routines. In general, to be threadsafe, a function should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED functions can be THREADSAFE.
- If the function is defined as NOT THREADSAFE, the database manager will never simultaneously invoke the function in the same process as another routine.

For FENCED functions, THREADSAFE is the default if the LANGUAGE is JAVA or CLR. For all other languages, NOT THREADSAFE is the default. If the function is defined with LANGUAGE OLE, THREADSAFE may not be specified (SQLSTATE 42613).

For NOT FENCED functions, THREADSAFE is the default. NOT THREADSAFE cannot be specified (SQLSTATE 42613).

RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT
This optional clause can be used to avoid a call to the external function if any of the arguments is null. If the user-defined function is defined to have no parameters, then this null argument condition cannot arise, and it does not matter how this specification is coded. If this clause is not specified, the default is RETURNS NULL ON NULL INPUT, except when PARAMETER STYLE JAVA is specified, in which case the default is CALLED ON NULL INPUT.

If RETURNS NULL ON NULL INPUT is specified, and if, at execution time, any one of the function's arguments is null, then the user-defined function is not called and the result is the null value.

If CALLED ON NULL INPUT is specified, then regardless of whether any arguments are null, the user-defined function is called. It can return a null value or a normal (non-null) value. But responsibility for testing for null argument values lies with the UDF.

The value NULL CALL may be used as a synonym for CALLED ON NULL INPUT for backwards and family compatibility. Similarly, NOT NULL CALL may be used as a synonym for RETURNS NULL ON NULL INPUT.

NO SQL, CONTAINS SQL, READS SQL DATA
Indicates whether the function issues any SQL statements and, if so, what type.

NO SQL
Indicates that the function cannot execute any SQL statements (SQLSTATE 38001).

CONTAINS SQL
Indicates that SQL statements that neither read nor modify SQL data can be executed by the function (SQLSTATE 38004 or 42985). Statements that are not supported in any function return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA
Indicates that some SQL statements that do not modify SQL data can be included in the function (SQLSTATE 38002 or 42985). Statements that are not supported in any function return a different error (SQLSTATE 38003 or 42985).

STATIC DISPATCH
This optional clause indicates that at function resolution time, DB2 chooses a function based on the static types (declared types) of the parameters of the function.

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the function takes an action that changes the state of an object that the database manager does not manage. An example of an external action is sending a message or writing a record to a file. The default is EXTERNAL ACTION.

EXTERNAL ACTION
Specifies that the function takes an action that changes the state of an object that the database manager does not manage.

A function with external actions might return incorrect results if the function is executed by parallel tasks. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead

of once for the function. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

NO EXTERNAL ACTION
Specifies that the function does not take any action that changes the state of an object that the database manager does not manage. The database manager uses this information during optimization of SQL statements.

NO SCRATCHPAD or SCRATCHPAD *length*
This optional clause may be used to specify whether a scratchpad is to be provided for an external function. (It is strongly recommended that user-defined functions be re-entrant, so a scratchpad provides a means for the function to "save state" from one call to the next.)

If SCRATCHPAD is specified, then at first invocation of the user-defined function, memory is allocated for a scratchpad to be used by the external function. This scratchpad has the following characteristics:

- *length*, if specified, sets the size of the scratchpad in bytes; this value must be between 1 and 32 767 (SQLSTATE 42820). The default size is 100 bytes.
- It is initialized to all X'00''s.
- Its scope is the SQL statement. There is one scratchpad per reference to the external function in the SQL statement. So if the UDFX function in the following statement is defined with the SCRATCHPAD keyword, three scratchpads would be assigned.

```
SELECT A, UDFX(A) FROM TABLEB
  WHERE UDFX(A) > 103 OR UDFX(A) < 19
```

  If ALLOW PARALLEL is specified or defaulted to, then the scope is different from the one shown previously. If the function is executed in multiple database partitions, a scratchpad would be assigned in each database partition where the function is processed, for each reference to the function in the SQL statement. Similarly, if the query is executed with intrapartition parallelism enabled, more than three scratchpads may be assigned.
- It is persistent. Its content is preserved from one external function call to the next. Any changes made to the scratchpad by the external function on one call will be there on the next call. The database manager initializes scratchpads at the beginning of execution of each SQL statement. The database manager may reset scratchpads at the beginning of execution of each subquery. The system issues a final call before resetting a scratchpad if the FINAL CALL option is specified.
- It can be used as a central point for system resources (for example, memory) which the external function might acquire. The function could acquire the memory on the first call, keep its address in the scratchpad, and refer to it in subsequent calls.

  (In such a case where system resource is acquired, the FINAL CALL keyword should also be specified; this causes a special call to be made at end-of-statement to allow the external function to free any system resources acquired.)

If SCRATCHPAD is specified, then on each invocation of the user-defined function an additional argument is passed to the external function which addresses the scratchpad.

If NO SCRATCHPAD is specified then no scratchpad is allocated or passed to the external function.

SCRATCHPAD is not supported for PARAMETER STYLE JAVA functions.

FINAL CALL or NO FINAL CALL

This optional clause specifies whether a final call is to be made to an external function. The purpose of such a final call is to enable the external function to free any system resources it has acquired. It can be useful in conjunction with the SCRATCHPAD keyword in situations where the external function acquires system resources such as memory and anchors them in the scratchpad. If FINAL CALL is specified, then at execution time:

- An additional argument is passed to the external function which specifies the type of call. The types of calls are:
  - Normal call: SQL arguments are passed and a result is expected to be returned.
  - First call: the first call to the external function for this reference to the user-defined function in this SQL statement. The first call is a normal call.
  - Final call: a final call to the external function to enable the function to free up resources. The final call is not a normal call. This final call occurs at the following times:
    - End-of-statement: This case occurs when the cursor is closed for cursor-oriented statements, or when the statement is through executing otherwise.
    - End-of-parallel-task: This case occurs when the function is executed by parallel tasks.
    - End-of-transaction or interrupt: This case occurs when the normal end-of-statement does not occur. For example, the logic of an application may for some reason bypass the close of the cursor. During this type of final call, no SQL statements may be issued except for CLOSE cursor (SQLSTATE 38505). This type of final call is indicated with a special value in the "call type" argument.

    If a commit operation occurs while a cursor defined as WITH HOLD is open, a final call is made at the subsequent close of the cursor or at the end of the application.

  If NO FINAL CALL is specified then no "call type" argument is passed to the external function, and no final call is made.

FINAL CALL is not supported for PARAMETER STYLE JAVA functions.

ALLOW PARALLEL or DISALLOW PARALLEL

This optional clause specifies whether, for a single reference to the function, the invocation of the function can be parallelized. In general, the invocations of most scalar functions should be parallelizable, but there may be functions (such as those depending on a single copy of a scratchpad) that cannot. If either ALLOW PARALLEL or DISALLOW PARALLEL are specified for a scalar function, then DB2 will accept this specification. The following questions should be considered in determining which keyword is appropriate for the function.

- Are all the UDF invocations completely independent of each other? If YES, then specify ALLOW PARALLEL.
- Does each UDF invocation update the scratchpad, providing value(s) that are of interest to the next invocation? (For example, the incrementing of a counter.) If YES, then specify DISALLOW PARALLEL or accept the default.
- Is there some external action performed by the UDF which should happen only on one database partition? If YES, then specify DISALLOW PARALLEL or accept the default.

- Is the scratchpad used, but only so that some expensive initialization processing can be performed a minimal number of times? If YES, then specify ALLOW PARALLEL.

In any case, the body of every external function should be in a directory that is available on every database partition.

The default value is ALLOW PARALLEL, except if one or more of the following options is specified in the statement.

- NOT DETERMINISTIC
- EXTERNAL ACTION
- SCRATCHPAD
- FINAL CALL

If any of these options is specified or implied, the default value is DISALLOW PARALLEL.

INHERIT SPECIAL REGISTERS

This optional clause specifies that updatable special registers in the function will inherit their initial values from the environment of the invoking statement. For a function invoked in the select-statement of a cursor, the initial values are inherited from the environment when the cursor is opened. For a routine invoked in a nested object (for example a trigger or view), the initial values are inherited from the runtime environment (not inherited from the object definition).

No changes to the special registers are passed back to the invoker of the function.

Non-updatable special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore set to their default values.

NO DBINFO or DBINFO

This optional clause specifies whether certain specific information known by DB2 will be passed to the UDF as an additional invocation-time argument (DBINFO) or not (NO DBINFO). NO DBINFO is the default. DBINFO is not supported for LANGUAGE OLE (SQLSTATE 42613) or PARAMETER STYLE JAVA.

If DBINFO is specified, then a structure is passed to the UDF which contains the following information:

- Data base name - the name of the currently connected database.
- Application ID - unique application ID which is established for each connection to the database.
- Application Authorization ID - the application runtime authorization ID, regardless of the nested UDFs in between this UDF and the application.
- Code page - identifies the database code page.
- Schema name - under the exact same conditions as for Table name, contains the name of the schema; otherwise blank.
- Table name - if and only if the UDF reference is either the right side of a SET clause in an UPDATE statement or an item in the VALUES list of an INSERT statement, contains the unqualified name of the table being updated or inserted; otherwise blank.
- Column name - under the exact same conditions as for Table name, contains the name of the column being updated or inserted; otherwise blank.

- Database version/release - identifies the version, release and modification level of the database server invoking the UDF.
- Platform - contains the server's platform type.
- Table function result column numbers - not applicable to external scalar functions.

TRANSFORM GROUP *group-name*
Indicates the transform group to be used for user-defined structured type transformations when invoking the function. A transform is required if the function definition includes a user-defined structured type as either a parameter or returns data type. If this clause is not specified, the default group name DB2_FUNCTION is used. If the specified (or default) *group-name* is not defined for a referenced structured type, an error is raised (SQLSTATE 42741). If a required FROM SQL or TO SQL transform function is not defined for the given group-name and structured type, an error is raised (SQLSTATE 42744).

The transform functions, both FROM SQL and TO SQL, whether designated or implied, must be SQL functions which properly transform between the structured type and its built in type attributes.

PREDICATES
Defines the filtering or index extension exploitation performed when this function is used in a predicate. A predicate-specification allows the optional SELECTIVITY clause of a search-condition to be specified. If the PREDICATES clause is specified, the function must be defined as DETERMINISTIC with NO EXTERNAL ACTION (SQLSTATE 42613). If the PREDICATES clause is specified, and the database is not a Unicode database, PARAMETER CCSID UNICODE must not be specified (SQLSTATE 42613).

WHEN *comparison-operator*
Introduces a specific use of the function in a predicate with a comparison operator ("=", "<", ">", ">=", "<=", "<>").

constant
Specifies a constant value with a data type comparable to the RETURNS type of the function (SQLSTATE 42818). When a predicate uses this function with the same comparison operator and this constant, the specified filtering and index exploitation will be considered by the optimizer.

EXPRESSION AS *expression-name*
Provides a name for an expression. When a predicate uses this function with the same comparison operator and an expression, filtering and index exploitation may be used. The expression is assigned an expression name so that it can be used as a search function argument. The *expression-name* cannot be the same as any *parameter-name* of the function being created (SQLSTATE 42711). When an expression is specified, the type of the expression is identified.

FILTER USING
Allows specification of an external function or a case expression to be used for additional filtering of the result table.

function-invocation
Specifies a filter function that can be used to perform additional filtering of the result table. This is a version of the defined function (used in the predicate) that reduces the number of rows on which the user-defined predicate must be executed, to determine if rows qualify. If the results produced by the index are close to the results expected

for the user-defined predicate, applying the filtering function may be redundant. If not specified, data filtering is not performed.

This function can use any *parameter-name*, the *expression-name*, or constants as arguments (SQLSTATE 42703), and returns an integer (SQLSTATE 428E4). A return value of 1 means the row is kept, otherwise it is discarded.

This function must also:

- Not be defined with LANGUAGE SQL (SQLSTATE 429B4)
- Not be defined with NOT DETERMINISTIC or EXTERNAL ACTION (SQLSTATE 42845)
- Not have a structured data type as the data type of any of the parameters (SQLSTATE 428E3)
- Not include a subquery (SQLSTATE 428E4)
- Not include an XMLQUERY or XMLEXISTS expression (SQLSTATE 428E4)

If an argument invokes another function or method, these rules are also enforced for this nested function or method. However, system-generated observer methods are allowed as arguments to the filter function (or any function or method used as an argument), as long as the argument evaluates to a built-in data type.

The definer of the function must have EXECUTE privilege on the specified filter function.

The *function-invocation* clause must not exceed 65 536 bytes in length in the database code page (SQLSTATE 22001).

case-expression
Specifies a case expression for additional filtering of the result table. The *searched-when-clause* and *simple-when-clause* can use *parameter-name*, *expression-name*, or a constant (SQLSTATE 42703). An external function with the rules specified in FILTER USING *function-invocation* may be used as a result-expression. Any function or method referenced in the *case-expression* must also conform to the four rules listed under *function-invocation*.

Subqueries and XMLQUERY or XMLEXISTS expressions cannot be used anywhere in the *case-expression* (SQLSTATE 428E4).

The case expression must return an integer (SQLSTATE 428E4). A return value of 1 in the result-expression means that the row is kept; otherwise it is discarded.

The *case-invocation* clause must not exceed 65 536 bytes in length in the database code page (SQLSTATE 22001).

index-exploitation
Defines a set of rules in terms of the search method of an index extension that can be used to exploit the index.

SEARCH BY INDEX EXTENSION *index-extension-name*
Identifies the index extension. The *index-extension-name* must identify an existing index extension.

EXACT
Indicates that the index lookup is exact in terms of the predicate evaluation. Use EXACT to tell DB2 that neither the original user-defined predicate function or the filter need to be applied after the

index lookup. The EXACT predicate is useful when the index lookup returns the same results as the predicate.

If EXACT is not specified, then the original user-defined predicate is applied after index lookup. If the index is expected to provide only an approximation of the predicate, do not specify the EXACT option.

If the index lookup is not used, then the filter function and the original predicate have to be applied.

exploitation-rule
Describes the search targets and search arguments and how they can be used to perform the index search through a search method defined in the index extension.

WHEN KEY (*parameter-name1***)**
This defines the search target. Only one search target can be specified for a key. The *parameter-name1* value identifies parameter names of the defined function (SQLSTATE 42703 or 428E8).

The data type of *parameter-name1* must match that of the source key specified in the index extension (SQLSTATE 428EY). The match must be exact for built-in and distinct data types and within the same structured type hierarchy for structured types.

This clause is true when the values of the named parameter are columns that are covered by an index based on the index extension specified.

USE *search-method-name***(***parameter-name2***,...)**
This defines the search argument. It identifies which search method to use from those defined in the index extension. The *search-method-name* must match a search method defined in the index extension (SQLSTATE 42743). The *parameter-name2* values identify parameter names of the defined function or the *expression-name* in the EXPRESSION AS clause (SQLSTATE 42703). It must be different from any parameter name specified in the search target (SQLSTATE 428E9). The number of parameters and the data type of each *parameter-name2* must match the parameters defined for the search method in the index extension (SQLSTATE 42816). The match must be exact for built-in and distinct data types and within the same structured type hierarchy for structured types.

NOT SECURED or SECURED
Specifies whether the function is considered secure for row and column access control. The default is NOT SECURED.

NOT SECURED
Indicates that the function is not considered secure. When the function is invoked, the arguments of the function must not reference a column for which a column mask is enabled and column level access control is activated for its table (SQLSTATE 428HA). This rule applies to the non secure user-defined functions that are invoked anywhere in the statement.

SECURED
Indicates that the function is considered secure. The function must be secure when it is referenced in a row permission or a column mask (SQLSTATE 428H8).

Notes

- Determining whether one data type is castable to another data type does not consider length or precision and scale for parameterized data types such as CHAR and DECIMAL. Therefore, errors may occur when using a function as a result of attempting to cast a value of the source data type to a value of the target data type. For example, VARCHAR is castable to DATE but if the source type is actually defined as VARCHAR(5), an error will occur when using the function.
- When choosing the data types for the parameters of a user-defined function, consider the rules for promotion that will affect its input values (see "Promotion of data types"). For example, a constant which may be used as an input value could have a built-in data type different from the one expected and, more significantly, may not be promoted to the data type expected. Based on the rules for promotion, it is generally recommended to use the following data types for parameters:
  - INTEGER instead of SMALLINT
  - DOUBLE instead of REAL
  - VARCHAR instead of CHAR
  - VARGRAPHIC instead of GRAPHIC
- For portability of UDFs across platforms the following data types should not be used:
  - FLOAT- use DOUBLE or REAL instead.
  - NUMERIC- use DECIMAL instead.
  - LONG VARCHAR- use CLOB (or BLOB) instead.
- A function and a method may not be in an overriding relationship (SQLSTATE 42745). For more information about overriding, see "CREATE TYPE (Structured)".
- A function may not have the same signature as a method (comparing the first *parameter-type* of the function with the *subject-type* of the method) (SQLSTATE 42723).
- Creating a function with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- In a partitioned database environment, the use of SQL in external user-defined functions or methods is not supported (SQLSTATE 42997).
- Only routines defined as NO SQL can be used to define an index extension (SQLSTATE 428F8).
- If the function allows SQL, the external program must not attempt to access any federated objects (SQLSTATE 55047).
- A Java routine defined as NOT FENCED will be invoked as if it had been defined as FENCED THREADSAFE.
- XML parameters are only supported in LANGUAGE JAVA external functions when the PARAMETER STYLE DB2GENERAL clause is specified.
- ***Table access restrictions***

  If a function is defined as READS SQL DATA, no statement in the function can access a table that is being modified by the statement which invoked the function (SQLSTATE 57053). For example, suppose the user-defined function BONUS() is defined as READS SQL DATA. If the statement UPDATE EMPLOYEE SET SALARY = SALARY + BONUS(EMPNO) is invoked, no SQL statement in the BONUS function can read from the EMPLOYEE table.

- ***Setting of the default value***: Parameters of a function that are defined with a default value are set to their default value when the functions is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the function is invoked.
- ***Privileges***: The definer of a function always receives the EXECUTE privilege WITH GRANT OPTION on the function, as well as the right to drop the function.

  When the function is used in an SQL statement, the function definer must have the EXECUTE privilege on any packages used by the function.
- ***EXTERNAL ACTION functions***: If an EXTERNAL ACTION function is invoked in other than the outermost select list, the results are unpredictable since the number of times the function is invoked will vary depending on the access plan used.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - PARAMETER STYLE DB2SQL can be specified in place of PARAMETER STYLE SQL
  - NOT VARIANT can be specified in place of DETERMINISTIC, and VARIANT can be specified in place of NOT DETERMINISTIC
  - NULL CALL can be specified in place of CALLED ON NULL INPUT, and NOT NULL CALL can be specified in place of RETURNS NULL ON NULL INPUT

  The following syntax is accepted as the default behavior:
  - ASUTIME NO LIMIT
  - NO COLLID
  - PROGRAM TYPE SUB
  - STAY RESIDENT NO
  - CCSID UNICODE in a Unicode database
  - CCSID ASCII in a non-Unicode database if PARAMETER CCSID UNICODE is not specified
- ***Creating a secure function***: Normally users with SECADM authority do not have privileges to create database objects such as triggers and functions. Typically; they will examine the data accessed by the function, ensure it is secure, then grant the CREATE_SECURE_OBJECT authority to someone who currently has required privileges to create a secure user-defined function. After the function is created, they will revoke the CREATE_SECURE_OBJECT authority from the function owner.

  The SECURED attribute is considered to be an assertion that declares the user has established a change control audit procedure for all changes to the user-defined function. The database manager assumes that such a control audit procedure is in place for all subsequent ALTER FUNCTION statements or changes to external packages.
- ***Invoking other user-defined functions in a secure function***: If a secure user-defined function invokes other user-defined functions, the database manager does not validate whether those nested user-defined functions have the SECURED attribute. If those nested functions can access sensitive data, the user with SECADM authority needs to ensure those functions are allowed to access those data and a change control audit procedure has been established for all changes to those functions.

- ***Replacing an existing function such that the secure attribute is changed (from SECURED to NOT SECURED and vice versa)***: Packages and dynamically cached SQL statements that depend on the function may be invalidated because the secure attribute affects the access path selection for statements involving tables for which row or column level access control is activated.

Examples

- *Example 1:* Pellow is registering the CENTRE function in his PELLOW schema. Let those keywords that will default do so, and let the system provide a function specific name:

```
CREATE FUNCTION CENTRE (INT,FLOAT)
  RETURNS FLOAT
  EXTERNAL NAME 'mod!middle'
  LANGUAGE C
  PARAMETER STYLE SQL
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
```

- *Example 2:* Now, McBride (who has DBADM authority) is registering another CENTRE function in the PELLOW schema, giving it an explicit specific name for subsequent data definition language use, and explicitly providing all keyword values. Note also that this function uses a scratchpad and presumably is accumulating data there that affects subsequent results. Since DISALLOW PARALLEL is specified, any reference to the function is not parallelized and therefore a single scratchpad is used to perform some one-time only initialization and save the results.

```
CREATE FUNCTION PELLOW.CENTRE (FLOAT, FLOAT, FLOAT)
  RETURNS DECIMAL(8,4) CAST FROM FLOAT
  SPECIFIC FOCUS92
  EXTERNAL NAME 'effects!focalpt'
  LANGUAGE C   PARAMETER STYLE SQL
  DETERMINISTIC   FENCED   NOT NULL CALL   NO SQL   NO EXTERNAL ACTION
  SCRATCHPAD   NO FINAL CALL
  DISALLOW PARALLEL
```

- *Example 3:* The following example is the C language user-defined function program written to implement the rule `output = 2 * input - 4` returning NULL if and only if the input is null. It could be written even more simply (that is, without null checking), if the CREATE FUNCTION statement had used NOT NULL CALL. The CREATE FUNCTION statement:

```
CREATE FUNCTION ntest1 (SMALLINT)
  RETURNS SMALLINT
  EXTERNAL NAME 'ntest1!nudft1'
  LANGUAGE C    PARAMETER STYLE SQL
  DETERMINISTIC   NOT FENCED   NULL CALL
  NO SQL   NO EXTERNAL ACTION
```

  The program code:

```
#include "sqlsystm.h"
/* NUDFT1 IS A USER_DEFINED SCALAR FUNCTION */
/* udft1 accepts smallint input
and produces smallint output
implementing the rule:
if (input is null)
set output = null;
else
set output = 2 * input - 4;
*/
void SQL_API_FN nudft1
(short *input,      /* ptr to input arg */
short *output,     /* ptr to where result goes */
short *input_ind,  /* ptr to input indicator var */
```

```
short *output_ind, /* ptr to output indicator var */
char sqlstate[6],  /* sqlstate, allows for null-term */
char fname[28],    /* fully qual func name, nul-term */
char finst[19],    /* func specific name,  null-term */
char msgtext[71])  /* msg text buffer,     null-term */
{
/* first test for null input */
if (*input_ind == -1)
{
/* input is null, likewise output */
*output_ind = -1;
}
else
{
/* input is not null.  set output to 2*input-4 */
*output = 2 * (*input) - 4;
/* and set out null indicator to zero */
*output_ind = 0;
}
/* signal successful completion by leaving sqlstate as is */
/* and exit */
return;
}
/* end of UDF: NUDFT1 */
```

- *Example 4:* The following example registers a Java UDF which returns the position of the first vowel in a string. The UDF is written in Java, is to be run fenced, and is the findvwl method of class javaUDFs.

```
CREATE FUNCTION findv ( CLOB(100K))
  RETURNS INTEGER
  FENCED
  LANGUAGE JAVA
  PARAMETER STYLE JAVA
  EXTERNAL NAME 'javaUDFs.findvwl'
  NO EXTERNAL ACTION
  CALLED ON NULL INPUT
  DETERMINISTIC
  NO SQL
```

- *Example 5:* This example outlines a user-defined predicate WITHIN that takes two parameters, g1 and g2, of type SHAPE as input:

```
CREATE FUNCTION within (g1 SHAPE, g2 SHAPE)
RETURNS INTEGER
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NOT FENCED
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'db2sefn!SDESpatilRelations'
PREDICATES
WHEN = 1
FILTER USING mbrOverlap(g1..xmin, g1..ymin, g1..xmax, g1..max,
g2..xmin, g2..ymin, g2..xmax, g2..ymax)
SEARCH BY INDEX EXTENSION gridIndex
WHEN KEY(g1) USE withinExplRule(g2)
WHEN KEY(g2) USE withinExplRule(g1)
```

  The description of the WITHIN function is similar to that of any user-defined function, but the following additions indicate that this function can be used in a user-defined predicate.

  – **PREDICATES WHEN = 1** indicates that when this function appears as

```
within(g1, g2) = 1
```

in the WHERE clause of a DML statement, the predicate is to be treated as a user-defined predicate and the index defined by the index extension *gridIndex* should be used to retrieve rows that satisfy this predicate. If a constant is specified, the constant specified during the DML statement has to match exactly the constant specified in the create index statement. This condition is provided mainly to cover Boolean expression where the result type is either a 1 or a 0. For other cases, the EXPRESSION clause is a better choice.

- **FILTER USING mbrOverlap** refers to a filtering function mbrOverlap, which is a cheaper version of the WITHIN predicate. In this example, the mbrOverlap function takes the minimum bounding rectangles as input and quickly determines if they overlap or not. If the minimum bounding rectangles of the two input shapes do not overlap, then g1 will not be contained with g2. Therefore the tuple can be safely discarded, avoiding the application of the expensive WITHIN predicate.
- The **SEARCH BY INDEX EXTENSION** clause indicates that combinations of index extension and search target can be used for this user-defined predicate.

- *Example 6:* This example outlines a user-defined predicate DISTANCE that takes two parameters, P1 and P2, of type POINT as input:

```
CREATE FUNCTION distance (P1 POINT, P2 POINT)
  RETURNS INTEGER
  LANGUAGE C
  PARAMETER STYLE SQL
  DETERMINISTIC
  NOT FENCED
  NO SQL
  NO EXTERNAL ACTION
  EXTERNAL NAME 'db2sefn!SDEDistances'
  PREDICATES
  WHEN > EXPRESSION AS distExpr
  SEARCH BY INDEX EXTENSION gridIndex
  WHEN KEY(P1) USE distanceGrRule(P2, distExpr)
  WHEN KEY(P2) USE distanceGrRule(P1, distExpr)
```

  The description of the DISTANCE function is similar to that of any user-defined function, but the following additions indicate that when this function is used in a predicate, that predicate is a user-defined predicate.

  - **PREDICATES WHEN > EXPRESSION AS distExpr** is another valid predicate specification. When an expression is specified in the WHEN clause, the result type of that expression is used for determining if the predicate is a user-defined predicate in the DML statement. For example:

```
SELECT T1.C1
  FROM T1, T2
  WHERE distance (T1.P1, T2.P1) > T2.C2
```

    The predicate specification distance takes two parameters as input and compares the results with T2.C2, which is of type INTEGER. Since only the data type of the right side expression matters, (as opposed to using a specific constant), it is better to choose the EXPRESSION clause in the CREATE FUNCTION DDL for specifying a wildcard as the comparison value.

    Alternatively, the following statement is also a valid user-defined predicate:

```
SELECT T1.C1
  FROM T1, T2
  WHERE distance(T1.P1, T2.P1) > distance (T1.P2, T2.P2)
```

    There is currently a restriction that only the right side is treated as the expression; the term on the left side is the user-defined function for the user-defined predicate.

– The **SEARCH BY INDEX EXTENSION** clause indicates that combinations of index extension and search target can be used for this user-defined-predicate. In the case of the distance function, the expression identified as distExpr is also one of the search arguments that is passed to the range-producer function (defined as part of the index extension). The expression identifier is used to define a name for the expression so that it is passed to the range-producer function as an argument.

CREATE FUNCTION (external table)

The CREATE FUNCTION (External Table) statement is used to register a user-defined external table function at the current server.

A *table function* can be used in the FROM clause of a SELECT, and returns a table to the SELECT by returning one row at a time.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CREATE_EXTERNAL_ROUTINE authority on the database and at least one of the following authorities:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the function does not exist
  - CREATEIN privilege on the schema, if the schema name of the function exists
- DBADM authority

Group privileges are not considered for any table or view specified in the CREATE FUNCTION statement.

To create a not-fenced function, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

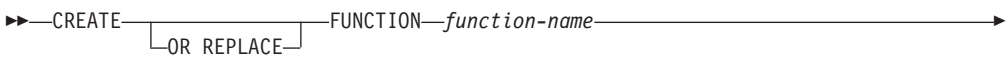
- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

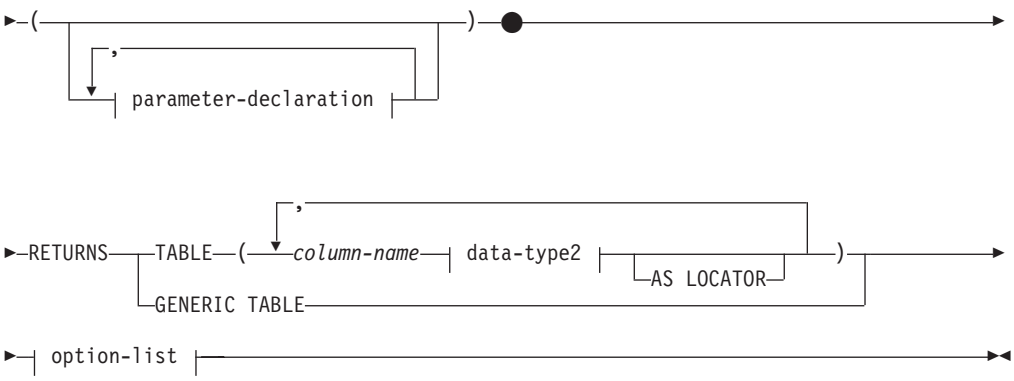
To create a fenced function, no additional authorities or privileges are required.

To replace an existing function, the authorization ID of the statement must be the owner of the existing function (SQLSTATE 42501).

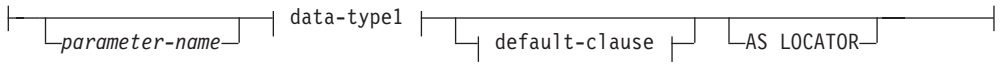
If the SECURED option is specified, the authorization ID of the statement must include SECADM or CREATE_SECURE_OBJECT authority (SQLSTATE 42501).

Syntax

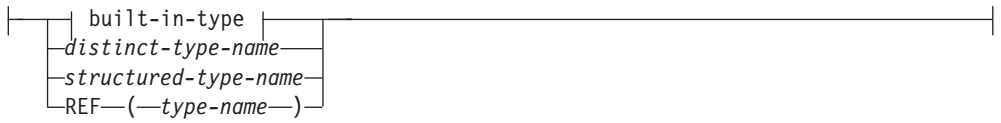




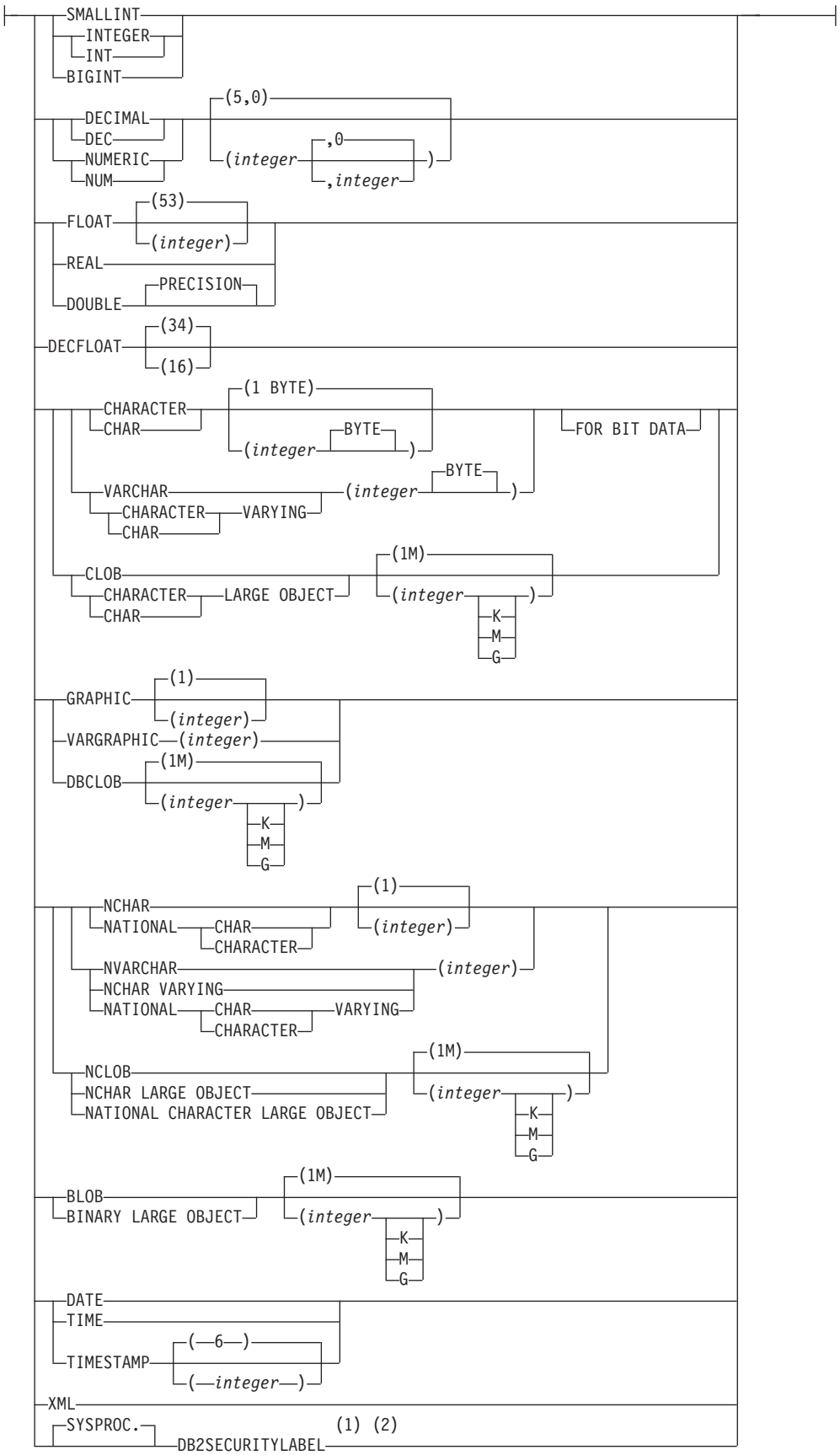
parameter-declaration:



data-type1, data-type2:

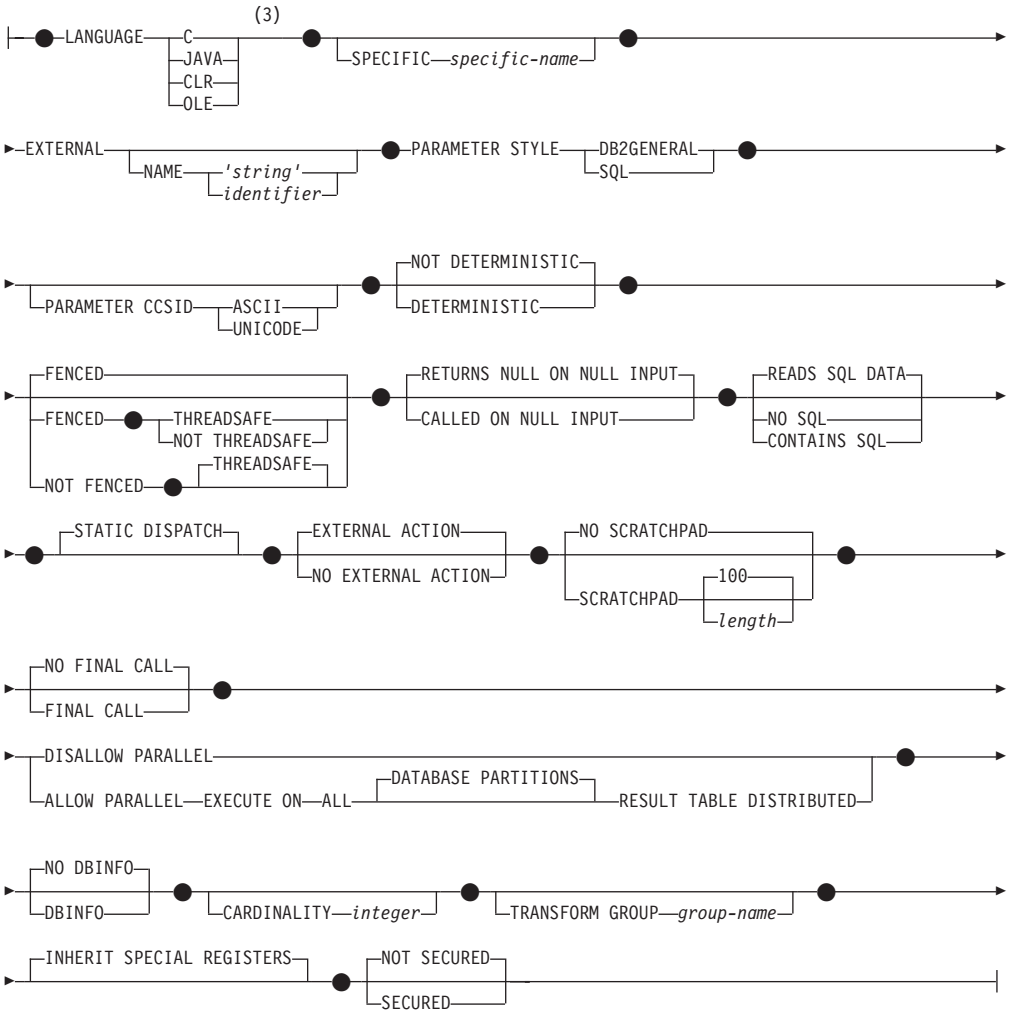


built-in type:


SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer)
,0
,integer
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
DECFLOAT
(34)
(16)
CHARACTER
CHAR
(1 BYTE)
(integer)
BYTE
FOR BIT DATA
VARCHAR
CHARACTER VARYING
CHAR VARYING
CLOB
CHARACTER LARGE OBJECT
CHAR LARGE OBJECT
(1M)
K
M
G
GRAPHIC
(1)
VARGRAPHIC
DBCLOB
NCHAR
NATIONAL CHAR
NATIONAL CHARACTER
NVARCHAR
NCHAR VARYING
NATIONAL CHAR VARYING
NATIONAL CHARACTER VARYING
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
BLOB
BINARY LARGE OBJECT
DATE
TIME
TIMESTAMP
(6)
XML
SYSPROC.
DB2SECURITYLABEL
(1) (2)

default-clause:

option-list:



Notes:

1 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

2 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.

3 For information about creating LANGUAGE OLE DB external table functions, see "CREATE FUNCTION (OLE DB External Table)". For information about creating LANGUAGE SQL table functions, see "CREATE FUNCTION (SQL Scalar, Table, or Row)".

Description

OR REPLACE
Specifies to replace the definition for the function if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the function are not affected. This option can be specified only by the owner of the object. This option is ignored if a definition for the function does not exist at the current server. To replace an existing function, the specific name and function name of the new definition must be the same as the specific name and function name of the old definition, or the signature of the new definition must match the signature of the old definition. Otherwise, a new function is created.

If the function is referenced in the definition of a row permission or a column mask, the function cannot be replaced (SQLSTATE 42893).

function-name
Names the function being defined. It is a qualified or unqualified name that designates a function. The unqualified form of *function-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier. The qualified name must not be the same as the data type of the first parameter, if that first parameter is a structured type.

The name, including the implicit or explicit qualifiers, together with the number of parameters and the data type of each parameter (without regard for any length, precision or scale attributes of the data type) must not identify a function described in the catalog (SQLSTATE 42723). The unqualified name, together with the number and data types of the parameters, while of course unique within its schema, need not be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *function-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

The same name can be used for more than one function if there is some difference in the signature of the functions. Although there is no prohibition against it, an external user-defined table function should not be given the same name as a built-in function.

(*parameter-declaration***,...)**
Identifies the number of input parameters of the function, and specifies the data type and optional default value of each parameter. One entry in the list must be specified for each parameter that the function will expect to receive. No more than 90 parameters are allowed (SQLSTATE 54023).

It is possible to register a function that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example:

```
CREATE FUNCTION WOOFER() ...
```

No two identically-named functions within a schema are permitted to have exactly the same type for all corresponding parameters. Lengths, precisions, and scales are not considered in this type comparison. Therefore, CHAR(8) and CHAR(35) are considered to be the same type, as are DECIMAL(11,2) and

DECIMAL (4,3). A weakly typed distinct type specified for a parameter is considered to be the same data type as the source type of the distinct type. For a Unicode database, CHAR(13) and GRAPHIC(8) are considered to be the same type. There is some further bundling of types that causes them to be treated as the same type for this purpose, such as DECIMAL and NUMERIC. A duplicate signature returns an error (SQLSTATE 42723).

parameter-name
Specifies an optional name for the input parameter. The name cannot be the same as any other *parameter-name* in the parameter list (SQLSTATE 42734).

data-type1
Specifies the data type of the input parameter. The data type can be a built-in data type, a distinct type, a structured type, or a reference type. For a more complete description of each built-in data type, see "CREATE TABLE". Some data types are not supported in all languages. For details on the mapping between SQL data types and host language data types, see "Data types that map to SQL data types in embedded SQL applications".

- A datetime type parameter is passed as a character data type, and the data is passed in the ISO format.
- DECIMAL (and NUMERIC) are invalid with LANGUAGE C and OLE (SQLSTATE 42815).
- XML is invalid with LANGUAGE OLE.
- Because the XML value that is seen inside a function is a serialized version of the XML value that is passed as a parameter in the function call, parameters of type XML must be declared using the syntax `XML AS CLOB(n)`.
- CLR does not support DECIMAL scale greater than 28 (SQLSTATE 42613).
- Array types cannot be specified (SQLSTATE 42815).

For a user-defined distinct type, the length, precision, or scale attributes for the parameter are those of the source type of the distinct type (those specified on CREATE TYPE). A distinct type parameter is passed as the source type of the distinct type. If the name of the distinct type is unqualified, the database manager resolves the schema name by searching the schemas in the SQL path.

For a user-defined structured type, the appropriate transform functions must exist in the associated transform group.

For a reference type, the parameter can be specified as REF(*type-name*) if the parameter is unscoped.

DEFAULT
Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression, or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see *default-clause* in the CREATE TABLE statement). Other special registers can be specified as the default by using an expression.

The expression can be any expression of the type described in "Expressions". If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified for a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB).

AS LOCATOR
Specifies that a locator to the value of the parameter is passed to the function instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB data type (SQLSTATE 42601). Passing locators instead of values can result in fewer bytes being passed to the function, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted, nor does it affect the function signature, which is used in function resolution.

If the function is FENCED and has the NO SQL option, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

RETURNS
Specifies the output of the function.

TABLE
Specifies that the output of the function is a table. The parentheses that follow this keyword delimit a list of the names and types of the columns of the table. The list style resembles the style of a simple CREATE TABLE statement which has no additional specifications (constraints, for example). No more than 255 columns are allowed (SQLSTATE 54011).

column-name
Specifies the name of this column. The name cannot be qualified and the same name cannot be used for more than one column of the table.

data-type2
Specifies the data type of the column, and can be any data type supported for a parameter of a UDF written in the particular language, except for structured types (SQLSTATE 42997).

AS LOCATOR
When *data-type2* is a LOB type or distinct type based on a LOB type, the use of this option indicates that the function is returning a locator for the LOB value that is instantiated in the result table.

The valid types for use with this clause are discussed in the "CREATE FUNCTION (external scalar)" statement topic.

GENERIC TABLE
Specifies that the output of the function is a generic table. This clause is allowed only if you specify the LANGUAGE JAVA clause and the PARAMETER STYLE DB2GENERAL clause (SQLSTATE 42613).

SPECIFIC *specific-name*
Provides a unique name for the instance of the function that is being defined. This specific name can be used when sourcing on this function, dropping the function, or commenting on the function. It can never be used to invoke the function. The unqualified form of *specific-name* is an SQL identifier. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifier, must not identify another function instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* may be the same as an existing *function-name*.

If no qualifier is specified, the qualifier that was used for *function-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *function-name* or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmmssxxx.

EXTERNAL

This clause indicates that the CREATE FUNCTION statement is being used to register a new function based on code written in an external programming language and adhering to the documented linkage conventions and interface.

If NAME clause is not specified "NAME *function-name*" is assumed.

NAME *'string'*

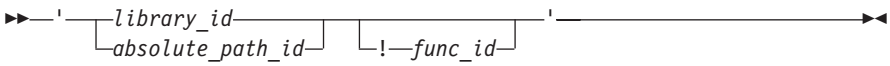
This clause identifies the user-written code that implements the function being defined.

The 'string' option is a string constant with a maximum of 254 bytes. The format used for the string is dependent on the LANGUAGE specified.

- For LANGUAGE C:

  The *string* specified is the library name and function within the library, which the database manager invokes to execute the user-defined function being created. The library (and the function within the library) do not need to exist when the CREATE FUNCTION statement is executed. However, when the function is used in an SQL statement, the library and function within the library must exist and be accessible from the database server machine.

  The *string* can be specified as follows:



  Extraneous blanks are not permitted within the single quotation marks.

  library_id

  Identifies the library name containing the function. The database manager will look for the library as follows:

  – On UNIX systems, if 'myfunc' was given as the *library_id*, and the database manager is being run from /u/production, the database manager will look for the function in library /u/production/sqllib/function/myfunc.

  – On Windows operating systems, the database manager will look for the function in a directory path that is specified by the LIBPATH or PATH environment variable.

  absolute_path_id

  Identifies the full path name of the file containing the function.

  On UNIX systems, for example, '/u/jchui/mylib/myfunc' would cause the database manager to look in /u/jchui/mylib for the myfunc shared library.

On Windows operating systems, 'd:\mylib\myfunc.dll' would cause the database manager to load the dynamic link library, myfunc.dll, from the d:\mylib directory. If an absolute path ID is being used to identify the routine body, be sure to append the .dll extension.

! func_id
Identifies the entry point name of the function to be invoked. The ! serves as a delimiter between the library ID and the function ID.

On a UNIX system, for example, 'mymod!func8' would direct the database manager to look for the library $inst_home_dir/sqllib/function/mymod and to use entry point func8 within that library.

On Windows operating systems, 'mymod!func8' would direct the database manager to load the mymod.dll file and to call the func8() function in the dynamic link library (DLL).

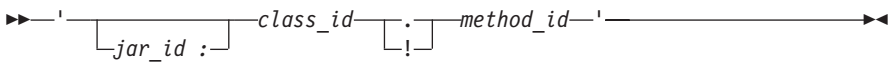
If the string is not properly formed, an error is returned (SQLSTATE 42878).

In any case, the body of every external function should be in a directory that is available on every database partition.

- For LANGUAGE JAVA:

  The *string* specified contains the optional jar file identifier, class identifier and method identifier, which the database manager invokes to execute the user-defined function being created. The class identifier and method identifier do not need to exist when the CREATE FUNCTION statement is executed. If a *jar_id* is specified, it must exist when the CREATE FUNCTION statement is executed. However, when the function is used in an SQL statement, the method identifier must exist and be accessible from the database server machine.

  The *string* can be specified as follows:



  Extraneous blanks are not permitted within the single quotation marks.

  jar_id
  Identifies the jar identifier given to the jar collection when it was installed in the database. It can be either a simple identifier, or a schema qualified identifier. Examples are 'myJar' and 'mySchema.myJar'

  class_id
  Identifies the class identifier of the Java object. If the class is part of a package, the class identifier part must include the complete package prefix, for example, 'myPacks.UserFuncs'. The Java virtual machine will look in directory '.../myPacks/UserFuncs/' for the classes. On Windows operating systems, the Java virtual machine will look in directory '...\myPacks\UserFuncs\'.

  method_id
  Identifies the method name of the Java object to be invoked.

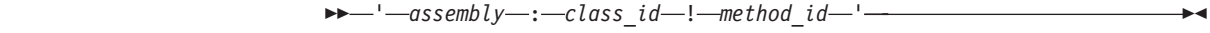
- For LANGUAGE CLR:

  The *string* specified represents the .NET assembly (library or executable), the class within that assembly, and the method within the class that the database manager invokes to execute the function being created. The module, class, and method do not need to exist when the CREATE

FUNCTION statement is executed. However, when the function is used in an SQL statement, the module, class, and method must exist and be accessible from the database server machine; otherwise, an error is returned (SQLSTATE 42724).

C++ routines that are compiled with the '/clr' compiler option to indicate that they include managed code extensions must be cataloged as 'LANGUAGE CLR' and not 'LANGUAGE C'. DB2 needs to know that the .NET infrastructure is being utilized in a user-defined function in order to make necessary runtime decisions. All user-defined functions using the .NET infrastructure must be cataloged as 'LANGUAGE CLR'.

The *string* can be specified as follows:



The name must be enclosed by single quotation marks. Extraneous blanks are not permitted.

assembly
Identifies the DLL or other assembly file in which the class resides. Any file extensions (such as .dll) must be specified. If the full path name is not given, the file must reside in the function directory of the DB2 install path (for example, c:\sqllib\function). If the file resides in a subdirectory of the install function directory, the subdirectory can be given before the file name rather than specifying the full path. For example, if your install directory is c:\sqllib and your assembly file is c:\sqllib\function\myprocs\mydotnet.dll, it is only necessary to specify 'myprocs\mydotnet.dll' for the assembly. The case sensitivity of this parameter is the same as the case sensitivity of the file system.

class_id
Specifies the name of the class within the given assembly in which the method that is to be invoked resides. If the class resides within a namespace, the full namespace must be given in addition to the class. For example, if the class EmployeeClass is in namespace MyCompany.ProcedureClasses, then MyCompany.ProcedureClasses.EmployeeClass must be specified for the class. Note that the compilers for some .NET languages will add the project name as a namespace for the class, and the behavior may differ depending on whether the command line compiler or the GUI compiler is used. This parameter is case sensitive.

method_id
Specifies the method within the given class that is to be invoked. This parameter is case sensitive.

- For LANGUAGE OLE:

  The *string* specified is the OLE programmatic identifier (progid) or class identifier (clsid), and method identifier, which the database manager invokes to execute the user-defined function being created. The programmatic identifier or class identifier, and method identifier do not need to exist when the CREATE FUNCTION statement is executed. However, when the function is used in an SQL statement, the method identifier must exist and be accessible from the database server machine; otherwise, an error is returned (SQLSTATE 42724).

  The *string* can be specified as follows:

Extraneous blanks are not permitted within the single quotation marks.

progid
Identifies the programmatic identifier of the OLE object.

progid is not interpreted by the database manager but only forwarded to the OLE APIs at run time. The specified OLE object must be creatable and support late binding (also called IDispatch-based binding).

clsid
Identifies the class identifier of the OLE object to create. It can be used as an alternative for specifying a *progid* in the case that an OLE object is not registered with a progid. The *clsid* has the form:

{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}

where 'n' is an alphanumeric character. *clsid* is not interpreted by the database manager but only forwarded to the OLE APIs at run time.

method_id
Identifies the method name of the OLE object to be invoked.

NAME *identifier*
This clause identifies the name of the user-written code which implements the function being defined. The *identifier* specified is an SQL identifier. The SQL identifier is used as the *library-id* in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C.

LANGUAGE
This mandatory clause is used to specify the language interface convention to which the user-defined function body is written.

C This means the database manager will call the user-defined function as if it were a C function. The user-defined function must conform to the C language calling and linkage convention as defined by the standard ANSI C prototype.

JAVA This means the database manager will call the user-defined function as a method in a Java class.

CLR This means the database manager will call the user-defined function as a method in a .NET class. At this time, LANGUAGE CLR is only supported for user-defined functions running on Windows operating systems. NOT FENCED cannot be specified for a CLR routine (SQLSTATE 42601).

OLE This means the database manager will call the user-defined function as if it were a method exposed by an OLE automation object. The user-defined function must conform with the OLE automation data types and invocation mechanism, as described in the *OLE Automation Programmer's Reference*.

LANGUAGE OLE is only supported for user-defined functions stored in DB2 for Windows 32-bit operating systems.

For information about creating LANGUAGE OLE DB external table functions, see "CREATE FUNCTION (OLE DB External Table)".

PARAMETER STYLE
This clause is used to specify the conventions used for passing parameters to and returning the value from functions.

DB2GENERAL
Used to specify the conventions for passing parameters to and returning the value from external functions that are defined as a method in a Java class. This can only be specified when LANGUAGE JAVA is used.

SQL
Used to specify the conventions for passing parameters to and returning the value from external functions that conform to C language calling and linkage conventions, methods exposed by OLE automation objects, or public static methods of a .NET object. This must be specified when LANGUAGE C, LANGUAGE CLR, or LANGUAGE OLE is used.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the function. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031). When the function is invoked, the application code page for the function is the database code page.

UNICODE
Specifies that string data is encoded in Unicode. If the database is a Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8. In either case, when the function is invoked, the application code page for the function is 1208.

If the database is not a Unicode database, and a function with PARAMETER CCSID UNICODE is created, the function cannot have any graphic types or user-defined types (SQLSTATE 560C1).

If the database is not a Unicode database, table functions can be created with PARAMETER CCSID UNICODE, but the following rules apply:

- The alternate collating sequence must be specified in the database configuration before creating the table function (SQLSTATE 56031). PARAMETER CCSID UNICODE table functions collate with the alternate collating sequence specified in the database configuration.
- Tables or table functions created with CCSID ASCII, and tables or table functions created with CCSID UNICODE, cannot both be used in a single SQL statement (SQLSTATE 53090). This applies to tables and table functions referenced directly in the statement, as well as to tables and table functions referenced indirectly (such as, for example, through referential integrity constraints, triggers, materialized query tables, and tables in the body of views).
- Table functions created with PARAMETER CCSID UNICODE cannot be referenced in SQL functions or SQL methods (SQLSTATE 560C0).
- An SQL statement that references a table function created with PARAMETER CCSID UNICODE cannot invoke an SQL function or SQL method (SQLSTATE 53090).

- Graphic types, the XML type, and user-defined types cannot be used as parameters to PARAMETER CCSID UNICODE table functions (SQLSTATE 560C1).
- Statements that reference a PARAMETER CCSID UNICODE table function can only be invoked from a DB2 Version 8.1 or later client (SQLSTATE 42997).
- SQL statements are always interpreted in the database code page. In particular, this means that every character in literals, hex literals, and delimited identifiers must have a representation in the database code page; otherwise, the character will be replaced with the substitution character.

If the database is not a Unicode database, and the alternate collating sequence has been specified in the database configuration, functions can be created with either PARAMETER CCSID ASCII or PARAMETER CCSID UNICODE. All string data passed into and out of the function will be converted to the appropriate code page.

This clause cannot be specified with LANGUAGE OLE, LANGUAGE JAVA, or LANGUAGE CLR (SQLSTATE 42613).

DETERMINISTIC or NOT DETERMINISTIC

This optional clause specifies whether the function always returns the same results for given argument values (DETERMINISTIC) or whether the function depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC function must always return the same table from successive invocations with identical inputs. Optimizations taking advantage of the fact that identical inputs always produce the same results are prevented by specifying NOT DETERMINISTIC. An example of a table function that is non-deterministic is one that references special registers, global variables, non-deterministic functions, or sequences in a way that affects the table function result table.

FENCED or NOT FENCED

This clause specifies whether or not the function is considered "safe" to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED).

If a function is registered as FENCED, the database manager protects its internal resources (for example, data buffers) from access by the function. Most functions will have the option of running as FENCED or NOT FENCED. In general, a function running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for functions not adequately coded, reviewed and tested can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED user defined functions are used.

Only FENCED can be specified for a function with LANGUAGE OLE or NOT THREADSAFE (SQLSTATE 42613).

If the function is FENCED and has the NO SQL option, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

Either SYSADM authority, DBADM authority, or a special authority (CREATE_NOT_FENCED_ROUTINE) is required to register a user-defined function as NOT FENCED.

LANGUAGE CLR user-defined functions cannot be created when specifying the NOT FENCED clause (SQLSTATE 42601).

THREADSAFE or NOT THREADSAFE

Specifies whether the function is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the function is defined with LANGUAGE other than OLE:

- If the function is defined as THREADSAFE, the database manager can invoke the function in the same process as other routines. In general, to be threadsafe, a function should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED functions can be THREADSAFE.
- If the function is defined as NOT THREADSAFE, the database manager will never simultaneously invoke the function in the same process as another routine.

For FENCED functions, THREADSAFE is the default if the LANGUAGE is JAVA or CLR. For all other languages, NOT THREADSAFE is the default. If the function is defined with LANGUAGE OLE, THREADSAFE may not be specified (SQLSTATE 42613).

For NOT FENCED functions, THREADSAFE is the default. NOT THREADSAFE cannot be specified (SQLSTATE 42613).

RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT

This optional clause may be used to avoid a call to the external function if any of the arguments is null. If the user-defined function is defined to have no parameters, then of course this null argument condition cannot arise, and it does not matter how this specification is coded.

If RETURNS NULL ON NULL INPUT is specified, and if, at table function OPEN time, any of the function's arguments are null, then the user-defined function is not called. The result of the attempted table function scan is the empty table (a table with no rows).

If CALLED ON NULL INPUT is specified, then regardless of whether any arguments are null, the user-defined function is called. It can return a null value or a normal (non-null) value. But responsibility for testing for null argument values lies with the UDF.

The value NULL CALL may be used as a synonym for CALLED ON NULL INPUT for backwards and family compatibility. Similarly, NOT NULL CALL may be used as a synonym for RETURNS NULL ON NULL INPUT.

NO SQL, CONTAINS SQL, READS SQL DATA

Indicates whether the function issues any SQL statements and, if so, what type.

NO SQL

Indicates that the function cannot execute any SQL statements (SQLSTATE 38001). If the ALLOW PARALLEL, EXECUTE ON ALL DATABASE PARTITIONS, and RESULT TABLE DISTRIBUTED clauses are all specified, NO SQL is the only option allowed.

CONTAINS SQL

Indicates that SQL statements that neither read nor modify SQL data can

be executed by the function (SQLSTATE 38004 or 42985). Statements that are not supported in any function return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA
Indicates that some SQL statements that do not modify SQL data can be included in the function (SQLSTATE 38002 or 42985). Statements that are not supported in any function return a different error (SQLSTATE 38003 or 42985).

STATIC DISPATCH
This optional clause indicates that at function resolution time, DB2 chooses a function based on the static types (declared types) of the parameters of the function.

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the function takes an action that changes the state of an object that the database manager does not manage. An example of an external action is sending a message or writing a record to a file. The default is EXTERNAL ACTION.

EXTERNAL ACTION
Specifies that the function takes an action that changes the state of an object that the database manager does not manage.

A function with external actions might return incorrect results if the function is executed by parallel tasks. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead of once for the function. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

NO EXTERNAL ACTION
Specifies that the function does not take any action that changes the state of an object that the database manager does not manage. The database manager uses this information during optimization of SQL statements.

NO SCRATCHPAD or SCRATCHPAD *length*
This optional clause may be used to specify whether a scratchpad is to be provided for an external function. (It is strongly recommended that user-defined functions be re-entrant, so a scratchpad provides a means for the function to "save state" from one call to the next.)

If SCRATCHPAD is specified, then at first invocation of the user-defined function, memory is allocated for a scratchpad to be used by the external function. This scratchpad has the following characteristics:

- *length*, if specified, sets the size of the scratchpad in bytes and must be between 1 and 32 767 (SQLSTATE 42820). The default value is 100.
- It is initialized to all X'00''s.
- Its scope is the SQL statement. There is one scratchpad per reference to the external function in the SQL statement. So if the UDFX function in the following statement is defined with the SCRATCHPAD keyword, two scratchpads would be assigned.

```
SELECT A.C1, B.C2
  FROM TABLE (UDFX(:hv1)) AS A,
       TABLE (UDFX(:hv1)) AS B
    WHERE ...
```

- It is persistent. It is initialized at the beginning of the execution of the statement, and can be used by the external table function to preserve the state of the scratchpad from one call to the next. If the FINAL CALL

keyword is also specified for the UDF, then the scratchpad is NEVER altered by DB2, and any resources anchored in the scratchpad should be released when the special FINAL call is made.

If NO FINAL CALL is specified or defaulted, then the external table function should clean up any such resources on the CLOSE call, as DB2 will re-initialize the scratchpad on each OPEN call. This determination of FINAL CALL or NO FINAL CALL and the associated behavior of the scratchpad could be an important consideration, particularly if the table function will be used in a subquery or join, since that is when multiple OPEN calls can occur during the execution of a statement.

- It can be used as a central point for system resources (for example, memory) which the external function might acquire. The function could acquire the memory on the first call, keep its address in the scratchpad, and refer to it in subsequent calls.

  (As previously outlined, the FINAL CALL/NO FINAL CALL keyword is used to control the re-initialization of the scratchpad, and also dictates when the external table function should release resources anchored in the scratchpad.)

If SCRATCHPAD is specified, then on each invocation of the user-defined function an additional argument is passed to the external function which addresses the scratchpad.

If NO SCRATCHPAD is specified then no scratchpad is allocated or passed to the external function.

FINAL CALL or NO FINAL CALL

This optional clause specifies whether a final call (and a separate first call) is to be made to an external function. It also controls when the scratchpad is re-initialized. If NO FINAL CALL is specified, then DB2 can only make three types of calls to the table function: open, fetch and close. However, if FINAL CALL is specified, then in addition to open, fetch and close, a first call and a final call can be made to the table function.

For external table functions, the call-type argument is ALWAYS present, regardless of which option is chosen.

If the final call is being made because of an interrupt or end-of-transaction, the UDF may not issue any SQL statements except for CLOSE cursor (SQLSTATE 38505). A special value is passed in the "call type" argument for these special final call situations.

DISALLOW PARALLEL or ALLOW PARALLEL EXECUTE ON ALL DATABASE PARTITIONS RESULT TABLE DISTRIBUTED

Specifies whether or not, for a single reference to the function, the invocation of the function is to be parallelized.

DISALLOW PARALLEL

Specifies that on each invocation of the function, DB2 invokes the function on a single database partition.

ALLOW PARALLEL EXECUTE ON ALL DATABASE PARTITIONS RESULT TABLE DISTRIBUTED

Specifies that on each invocation of the function, DB2 invokes the function on all database partitions. The union of the result sets obtained on each database partition is returned. The function cannot execute SQL statements (the NO SQL clause must also be specified).

NO DBINFO or DBINFO

This optional clause specifies whether certain specific information known to

DB2 is to be passed to the function as an additional invocation-time argument (DBINFO) or not (NO DBINFO). NO DBINFO is the default. DBINFO is not supported for LANGUAGE OLE (SQLSTATE 42613).

If DBINFO is specified, a structure containing the following information is passed to the function:

- Database name - the name of the currently connected database
- Application ID - the unique application ID that is established for each connection to the database
- Application authorization ID - the application runtime authorization ID, regardless of any nested functions between this function and the application
- Code page - the database code page
- Schema name - not applicable to external table functions
- Table name - not applicable to external table functions
- Column name - not applicable to external table functions
- Database version or release - the version, release, and modification level of the database server that is invoking the function
- Platform - the server's platform type
- Table function result column numbers - an array of result column numbers that is used by the statement referencing the function; this information enables the function to return only required column values instead of all column values
- Database partition number - the number of the database partition on which the external table function is invoked; in a single database partition environment, this value is 0

CARDINALITY *integer*

This optional clause provides an estimate of the expected number of rows to be returned by the function for optimization purposes. Valid values for *integer* range from 0 to 9 223 372 036 854 775 807 inclusive.

If the CARDINALITY clause is not specified for a table function, DB2 will assume a finite value as a default- the same value assumed for tables for which the RUNSTATS utility has not gathered statistics.

Warning: If a function does, in fact, have infinite cardinality - that is, it returns a row every time it is called to do so, and never returns the "end-of-table" condition - then queries that require the end-of-table condition to correctly function will be infinite, and will have to be interrupted. Examples of such queries are those that contain a GROUP BY or an ORDER BY clause. Writing such UDFs is not recommended.

TRANSFORM GROUP *group-name*

Indicates the transform group to be used for user-defined structured type transformations when invoking the function. A transform is required if the function definition includes a user-defined structured type as a parameter data type. If this clause is not specified, the default group name DB2_FUNCTION is used. If the specified (or default) *group-name* is not defined for a referenced structured type, an error results (SQLSTATE 42741). If a required FROM SQL transform function is not defined for the given group-name and structured type, an error results (SQLSTATE 42744).

INHERIT SPECIAL REGISTERS

This optional clause specifies that updatable special registers in the function will inherit their initial values from the environment of the invoking statement. For a function invoked in the select-statement of a cursor, the initial values are

inherited from the environment when the cursor is opened. For a routine invoked in a nested object (for example a trigger or view), the initial values are inherited from the runtime environment (not inherited from the object definition).

No changes to the special registers are passed back to the invoker of the function.

Non-updatable special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore set to their default values.

NOT SECURED or SECURED
Specifies whether the function is considered secure for row and column access control. The default is NOT SECURED.

NOT SECURED
Indicates that the function is not considered secure. When the function is invoked, the arguments of the function must not reference a column for which a column mask is enabled and column level access control is activated for its table (SQLSTATE 428HA). This rule applies to the non secure user-defined functions that are invoked anywhere in the statement.

SECURED
Indicates that the function is considered secure. The function must be secure when it is referenced in a row permission or a column mask (SQLSTATE 428H8, SQLCODE -20470).

Rules

- In a partitioned database environment, the use of SQL in external user-defined functions or methods is not supported (SQLSTATE 42997).
- Only routines defined as NO SQL can be used to define an index extension (SQLSTATE 428F8).
- If the function allows SQL, the external program must not attempt to access any federated objects (SQLSTATE 55047).
- ***Table access restrictions*** If a function is defined as READS SQL DATA, no statement in the function can access a table that is being modified by the statement which invoked the function (SQLSTATE 57053). For example, suppose the user-defined function BONUS() is defined as READS SQL DATA. If the statement UPDATE EMPLOYEE SET SALARY = SALARY + BONUS(EMPNO) is invoked, no SQL statement in the BONUS function can read from the EMPLOYEE table.

Notes

- When choosing the data types for the parameters of a user-defined function, consider the rules for promotion that will affect its input values. For example, a constant which may be used as an input value could have a built-in data type that is different from the one expected and, more significantly, may not be promoted to the data type expected. Based on the rules for promotion, it is generally recommended to use the following data types for parameters:
  - INTEGER instead of SMALLINT
  - DOUBLE instead of REAL
  - VARCHAR instead of CHAR
  - VARGRAPHIC instead of GRAPHIC
- For portability of UDFs across platforms, it is recommended to use the following data types:

  - DOUBLE or REAL instead of FLOAT
  - DECIMAL instead of NUMERIC
  - CLOB (or BLOB) instead of LONG VARCHAR
- Creating a function with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- A Java routine defined as NOT FENCED will be invoked as if it had been defined as FENCED THREADSAFE.
- ***Privileges***: The definer of a function always receives the EXECUTE privilege WITH GRANT OPTION on the function, as well as the right to drop the function. When the function is used in an SQL statement, the function definer must have the EXECUTE privilege on any packages used by the function.
- ***Setting of the default value***: Parameters of a function that are defined with a default value are set to their default value when the functions is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the function is invoked.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - PARAMETER STYLE DB2SQL can be specified in place of PARAMETER STYLE SQL
  - NOT VARIANT can be specified in place of DETERMINISTIC
  - VARIANT can be specified in place of NOT DETERMINISTIC
  - NULL CALL can be specified in place of CALLED ON NULL INPUT
  - NOT NULL CALL can be specified in place of RETURNS NULL ON NULL INPUT
  - DB2GENRL can be specified in place of DB2GENERAL

  The following syntax is accepted as the default behavior:
  - ASUTIME NO LIMIT
  - NO COLLID
  - PROGRAM TYPE SUB
  - STAY RESIDENT NO
  - CCSID UNICODE in a Unicode database
  - CCSID ASCII in a non-Unicode database if PARAMETER CCSID UNICODE is not specified
- ***Creating a secure function***: Normally users with SECADM authority do not have privileges to create database objects such as triggers and functions. Typically they will examine the data accessed by the function, ensure it is secure, then grant the CREATE_SECURE_OBJECT authority to someone who currently has required privileges to create a secure user-defined function. After the function is created, they will revoke the CREATE_SECURE_OBJECT authority from the function owner.

  The SECURED attribute is considered to be an assertion that declares the user has established a change control audit procedure for all changes to the user-defined function. The database manager assumes that such a control audit procedure is in place for all subsequent ALTER FUNCTION statements or changes to external packages.
- ***Invoking other user-defined functions in a secure function***: If a secure user-defined function invokes other user-defined functions, the database

manager does not validate whether those nested user-defined functions have the SECURED attribute. If those nested functions can access sensitive data, the user with SECADM authority needs to ensure those functions are allowed to access those data and a change control audit procedure has been established for all changes to those functions.

- ***Replacing an existing function such that the secure attribute is changed (from SECURED to NOT SECURED and vice versa)***: Packages and dynamically cached SQL statements that depend on the function may be invalidated because the secure attribute affects the access path selection for statements involving tables for which row or column level access control is activated.
- ***EXTERNAL ACTION functions***: If an EXTERNAL ACTION function is invoked in other than the outermost select list, the results are unpredictable since the number of times the function is invoked will vary depending on the access plan used.

Examples

- *Example 1:* The following example registers a table function written to return a row consisting of a single document identifier column for each known document in a text management system. The first parameter matches a given subject area and the second parameter contains a given string.

  Within the context of a single session, the UDF will always return the same table, and therefore it is defined as DETERMINISTIC. Note the RETURNS clause which defines the output from DOCMATCH. FINAL CALL must be specified for each table function. In addition, the DISALLOW PARALLEL keyword is added as table functions cannot operate in parallel. Although the size of the output for DOCMATCH is highly variable, CARDINALITY 20 is a representative value, and is specified to help the DB2 optimizer.

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
  RETURNS TABLE (DOC_ID CHAR(16))
  EXTERNAL NAME '/common/docfuncs/rajiv/udfmatch'
  LANGUAGE C
  PARAMETER STYLE SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  NOT FENCED
  SCRATCHPAD
  FINAL CALL
  DISALLOW PARALLEL
  CARDINALITY 20
```

- *Example 2:* The following example registers an OLE table function that is used to retrieve message header information and the partial message text of messages in Microsoft Exchange.

```
CREATE FUNCTION MAIL()
  RETURNS TABLE (TIMERECEIVED DATE,
                 SUBJECT VARCHAR(15),
                 SIZE INTEGER,
                 TEXT VARCHAR(30))
  EXTERNAL NAME 'tfmail.header!list'
  LANGUAGE OLE
  PARAMETER STYLE SQL
  NOT DETERMINISTIC
  FENCED
  CALLED ON NULL INPUT
  SCRATCHPAD
  FINAL CALL
  NO SQL
  EXTERNAL ACTION
  DISALLOW PARALLEL
```

CREATE FUNCTION (OLE DB external table)

The CREATE FUNCTION (OLE DB External Table) statement is used to register a user-defined OLE DB external table function to access data from an OLE DB provider.

A *table function* can be used in the FROM clause of a SELECT.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

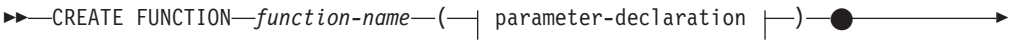
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

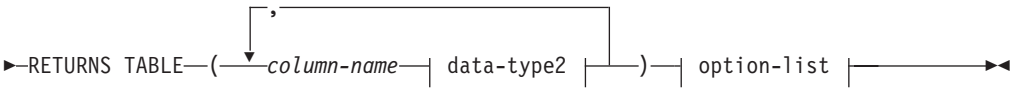
- CREATE_EXTERNAL_ROUTINE authority on the database and at least one of the following authorities:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the function does not exist
  - CREATEIN privilege on the schema, if the schema name of the function exists
- DBADM authority

Group privileges are not considered for any table or view specified in the CREATE FUNCTION statement.

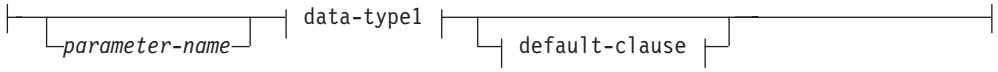
If the SECURED option is specified, the authorization ID of the statement must include SECADM authority or CREATE_SECURE_OBJECT authority (SQLSTATE 42501).

Syntax



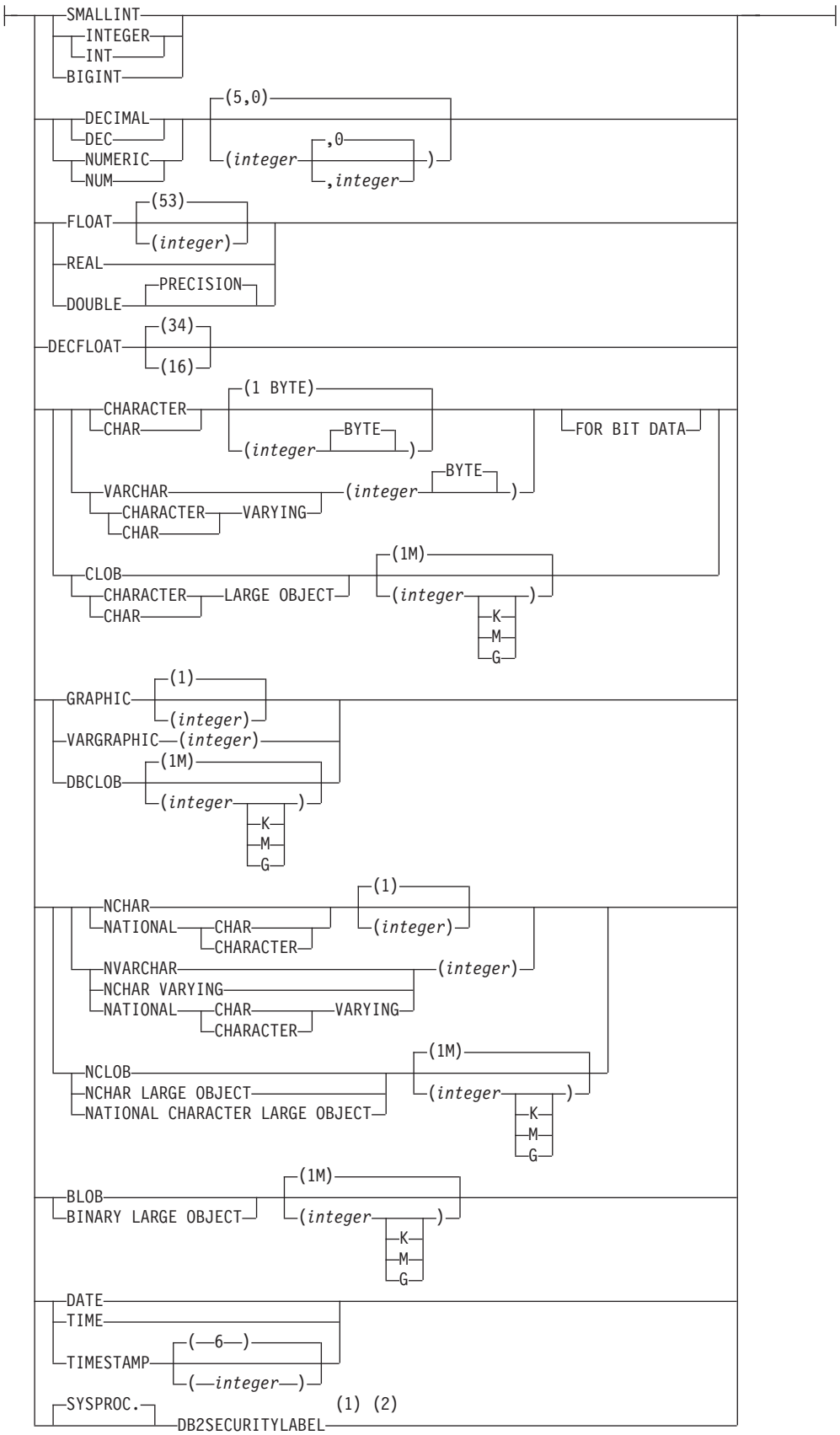


parameter-declaration:



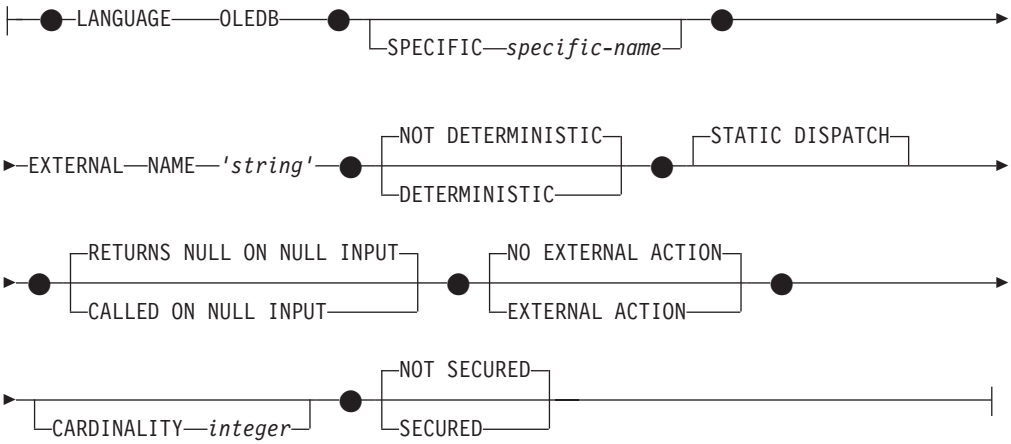
data-type1, data-type2:

built-in type:


SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer
,0
,integer)
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
DECFLOAT
(34)
(16)
CHARACTER
CHAR
(1 BYTE)
(integer
BYTE)
FOR BIT DATA
VARCHAR
CHARACTER
VARYING
CHAR
(integer
BYTE)
CLOB
CHARACTER
LARGE OBJECT
CHAR
(1M)
(integer
K
M
G)
GRAPHIC
(1)
(integer)
VARGRAPHIC
(integer)
DBCLOB
(1M)
(integer
K
M
G)
NCHAR
NATIONAL
CHAR
CHARACTER
(1)
(integer)
NVARCHAR
NCHAR VARYING
NATIONAL
CHAR
CHARACTER
VARYING
(integer)
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
(1M)
(integer
K
M
G)
BLOB
BINARY LARGE OBJECT
(1M)
(integer
K
M
G)
DATE
TIME
TIMESTAMP
(6)
(integer)
SYSPROC.
DB2SECURITYLABEL
(1) (2)

default-clause:

option-list:



Notes:

1 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

2 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.

Description

function-name

Names the function being defined. It is a qualified or unqualified name that designates a function. The unqualified form of *function-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters and the data type of each parameter (without regard for any length, precision or scale attributes of the data type) must not identify a function described in the catalog (SQLSTATE 42723). The unqualified name, together with the number and data types of the parameters, while of course unique within its schema, need not be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *function-name* (SQLSTATE 42939). The names are

SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

The same name can be used for more than one function if there is some difference in the signature of the functions. Although there is no prohibition against it, an external user-defined table function should not be given the same name as a built-in function.

(*parameter-declaration***,...)**
Identifies the number of input parameters of the function, and specifies the data type and optional default value of each parameter. If no input parameter is specified, data is retrieved from the external source possibly subsetted through query optimization. The input parameter passes command text to an OLE DB provider.

It is possible to register a function that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example:

```
CREATE FUNCTION WOOFER() ...
```

No two identically-named functions within a schema are permitted to have exactly the same type for all corresponding parameters. Lengths, precisions, and scales are not considered in this type comparison. Therefore, CHAR(8) and CHAR(35) are considered to be the same type. A weakly typed distinct type specified for a parameter is considered to be the same data type as the source type of the distinct type. For a Unicode database, CHAR(13) and GRAPHIC(8) are considered to be the same type. A duplicate signature returns an error (SQLSTATE 42723).

parameter-name
Specifies an optional name for the input parameter.

data-type1
Specifies the data type of the input parameter. The data type can be any character or graphic string data type or a distinct type based on a character or graphic string data type. Parameters of type XML are not supported (SQLSTATE 42815).

For a more complete description of each built-in data type, see "CREATE TABLE".

For a user-defined distinct type, the length, precision, or scale attributes for the parameter are those of the source type of the distinct type (those specified on CREATE TYPE). A distinct type parameter is passed as the source type of the distinct type. If the name of the distinct type is unqualified, the database manager resolves the schema name by searching the schemas in the SQL path.

DEFAULT
Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression, or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see *default-clause* in the CREATE TABLE statement). Other special registers can be specified as the default by using an expression.

The expression can be any expression of the type described in "Expressions". If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified for a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB).

RETURNS TABLE

Specifies that the output of the function is a table. The parentheses that follow this keyword delimit a list of the names and types of the columns of the table, resembling the style of a simple CREATE TABLE statement which has no additional specifications (constraints, for example).

column-name

Specifies the name of the column which must be the same as the corresponding rowset column name. The name cannot be qualified and the same name cannot be used for more than one column of the table.

data-type2

Specifies the data type of the column. XML is invalid (SQLSTATE 42815).

SPECIFIC *specific-name*

Provides a unique name for the instance of the function that is being defined. This specific name can be used when sourcing on this function, dropping the function, or commenting on the function. It can never be used to invoke the function. The unqualified form of *specific-name* is an SQL identifier. The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another function instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* may be the same as an existing *function-name*.

If no qualifier is specified, the qualifier that was used for *function-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *function-name* or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmmssxxx.

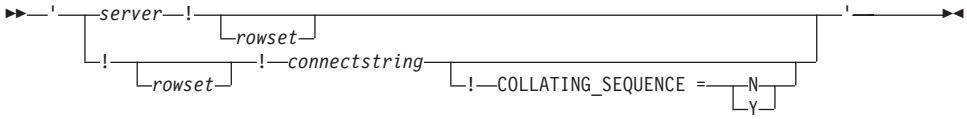
EXTERNAL NAME *'string'*

This clause identifies the external table and an OLE DB provider.

The 'string' option is a string constant with a maximum of 254 bytes.

The string specified is used to establish a connection and session with an OLE DB provider, and retrieve data from a rowset. The OLE DB provider and data source do not need to exist when the CREATE FUNCTION statement is executed.

The *string* can be specified as follows:



server

Identifies the local name of a data source as defined by “CREATE SERVER”.

rowset
Identifies the rowset (table) exposed by the OLE DB provider. Fully qualified table names must be provided for OLE DB providers that support catalog or schema names.

connectstring
String version of the initialization properties needed to connect to a data source. The basic format of a connection string is based on the ODBC connection string. The string contains a series of keyword/value pairs separated by semicolons. The equal sign (=) separates each keyword and its value. Keywords are the descriptions of the OLE DB initialization properties (property set DBPROPSET_DBINIT) or provider-specific keywords.

COLLATING_SEQUENCE
Specifies whether the data source uses the same collating sequence as DB2 Database for Linux, UNIX, and Windows. For details, see "CREATE SERVER". Valid values are as follows:

- Y = Same collating sequence
- N = Different collating sequence

If COLLATING_SEQUENCE is not specified, the data source is assumed to have a different collating sequence than DB2 Database for Linux, UNIX, and Windows.

If *server* is provided, *connectstring* or COLLATING_SEQUENCE are not allowed in the external name. They are defined as server options CONNECTSTRING and COLLATING_SEQUENCE. If no *server* is provided, a *connectstring* must be provided. If *rowset* is not provided, the table function must have an input parameter to pass through command text to the OLE DB provider.

LANGUAGE OLEDB
This means the database manager will deploy a built-in generic OLE DB consumer to retrieve data from the OLE DB provider. No table function implementation is required by the developer.

LANGUAGE OLEDB table functions can be created on any platform, but only executed on platforms supported by Microsoft OLE DB.

DETERMINISTIC or NOT DETERMINISTIC
This optional clause specifies whether the function always returns the same results for given argument values (DETERMINISTIC) or whether the function depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC function must always return the same table from successive invocations with identical inputs. Optimizations taking advantage of the fact that identical inputs always produce the same results are prevented by specifying NOT DETERMINISTIC.

STATIC DISPATCH
This optional clause indicates that at function resolution time, DB2 chooses a function based on the static types (declared types) of the parameters of the function.

RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT
This optional clause may be used to avoid a call to the external function if any of the arguments is null. If the user-defined function is defined to have no parameters, then of course this null argument condition cannot arise.

If RETURNS NULL ON NULL INPUT is specified and if at execution time any one of the function's arguments is null, the user-defined function is not called and the result is the empty table; that is, a table with no rows.

If CALLED ON NULL INPUT is specified, then at execution time regardless of whether any arguments are null, the user-defined function is called. It can return an empty table or not, depending on its logic. But responsibility for testing for null argument values lies with the UDF.

The value NULL CALL may be used as a synonym for CALLED ON NULL INPUT for backwards and family compatibility. Similarly, NOT NULL CALL may be used as a synonym for RETURNS NULL ON NULL INPUT.

NO EXTERNAL ACTION or EXTERNAL ACTION
Specifies whether the function takes an action that changes the state of an object that the database manager does not manage. An example of an external action is sending a message or writing a record to a file. The default is NO EXTERNAL ACTION.

NO EXTERNAL ACTION
Specifies that the function does not take any action that changes the state of an object that the database manager does not manage. The database manager uses this information during optimization of SQL statements.

EXTERNAL ACTION
Specifies that the function takes an action that changes the state of an object that the database manager does not manage.

CARDINALITY *integer*
This optional clause provides an estimate of the expected number of rows to be returned by the function for optimization purposes. Valid values for *integer* range from 0 to 2 147 483 647 inclusive.

If the CARDINALITY clause is not specified for a table function, DB2 will assume a finite value as a default- the same value assumed for tables for which the RUNSTATS utility has not gathered statistics.

Warning: If a function does, in fact, have infinite cardinality - that is, it returns a row every time it is called to do so, and never returns the "end-of-table" condition - then queries that require the end-of-table condition to correctly function will be infinite, and will have to be interrupted. Examples of such queries are those that contain a GROUP BY or an ORDER BY clause. Writing such UDFs is not recommended.

NOT SECURED or SECURED
Specifies whether the function is considered secure for row and column access control. The default is NOT SECURED.

NOT SECURED
Indicates that the function is not considered secure. When the function is invoked, the arguments of the function must not reference a column for which a column mask is enabled and column level access control is activated for its table (SQLSTATE 428HA). This rule applies to the non secure user-defined functions that are invoked anywhere in the statement.

SECURED
Indicates that the function is considered secure. The function must be secure when it is referenced in a row permission or a column mask (SQLSTATE 428H8).

Notes

- FENCED, FINAL CALL, SCRATCHPAD, PARAMETER STYLE SQL, DISALLOW PARALLEL, NO DBINFO, NOT THREADSAFE, and NO SQL are implicit in the statement and can be specified.
- When choosing the data types for the parameters of a user-defined function, consider the rules for promotion that will affect its input values. For example, a constant which may be used as an input value could have a built-in data type that is different from the one expected and, more significantly, may not be promoted to the data type expected. Based on the rules for promotion, it is generally recommended to use the following data types for parameters:
  - VARCHAR instead of CHAR
  - VARGRAPHIC instead of GRAPHIC
- For portability of UDFs across platforms, it is recommended to use the following data types:
  - DOUBLE or REAL instead of FLOAT
  - DECIMAL instead of NUMERIC
  - CLOB (or BLOB) instead of LONG VARCHAR
- Creating a function with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- ***Privileges***: The definer of a function always receives the EXECUTE privilege WITH GRANT OPTION on the function, as well as the right to drop the function.
- ***Setting of the default value***: Parameters of a function that are defined with a default value are set to their default value when the functions is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the function is invoked.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NOT VARIANT can be specified in place of DETERMINISTIC
  - VARIANT can be specified in place of NOT DETERMINISTIC
  - NULL CALL can be specified in place of CALLED ON NULL INPUT
  - NOT NULL CALL can be specified in place of RETURNS NULL ON NULL INPUT
- ***Creating a secure function***: Normally users with SECADM authority do not have privileges to create database objects such as triggers or functions. Typically they will examine the data accessed by the function, ensure it is secure, then grant the CREATE_SECURE_OBJECT authority to someone who currently has required privileges to create a secure user-defined function. After the function is created, they will revoke the CREATE_SECURE_OBJECT authority from the function owner.

  The SECURED attribute is considered to be an assertion that declares the user has established a change control audit procedure for all changes to the user-defined function. The database manager assumes that such a control audit procedure is in place for all subsequent ALTER FUNCTION statements or changes to external packages.
- ***Invoking other user-defined functions in a secure function***: If a secure user-defined function invokes other user-defined functions, the database manager does not validate whether those nested user-defined functions have the

SECURED attribute. If those nested functions can access sensitive data, the user with SECADM authority needs to ensure those functions are allowed to access those data and a change control audit procedure has been established for all changes to those functions.

- ***EXTERNAL ACTION functions***: If an EXTERNAL ACTION function is invoked in other than the outermost select list, the results are unpredictable since the number of times the function is invoked will vary depending on the access plan used.

Examples

- *Example 1:* Register an OLE DB table function, which retrieves order information from a Microsoft Access database. The connection string is defined in the external name.

```
CREATE FUNCTION orders ()
  RETURNS TABLE (orderid INTEGER,
                 customerid CHAR(5),
                 employeeid INTEGER,
                 orderdate TIMESTAMP,
                 requireddate TIMESTAMP,
                 shippeddate TIMESTAMP,
                 shipvia INTEGER,
                 freight dec(19,4))
  LANGUAGE OLEDB
  EXTERNAL NAME '!orders!Provider=Microsoft.Jet.OLEDB.3.51;
                         Data Source=c:\sqllib\samples\oledb\nwind.mdb
  !COLLATING_SEQUENCE=Y';
```

- *Example 2:* Register an OLE DB table function, which retrieves customer information from an Oracle database. The connection string is provided through a server definition. The table name is fully qualified in the external name. The local user john is mapped to the remote user dave. Other users will use the guest user ID in the connection string.

```
CREATE SERVER spirit
  WRAPPER OLEDB
  OPTIONS (CONNECTSTRING 'Provider=MSDAORA;Persist Security Info=False;
                         User ID=guest;password=pwd;Locale Identifier=1033;
                         OLE DB Services=CLIENTCURSOR;Data Source=spirit');

CREATE USER MAPPING FOR john
  SERVER spirit
  OPTIONS (REMOTE_AUTHID 'dave', REMOTE_PASSWORD 'mypwd');

CREATE FUNCTION customers ()
  RETURNS TABLE (customer_id INTEGER,
                 name  VARCHAR(20),
                 address VARCHAR(20),
                 city VARCHAR(20),
                 state VARCHAR(5),
                 zip_code INTEGER)
  LANGUAGE OLEDB
  EXTERNAL NAME 'spirit!demo.customer';
```

- *Example 3:* Register an OLE DB table function, which retrieves information about stores from a MS SQL Server 7.0 database. The connection string is provided in the external name. The table function has an input parameter to pass through command text to the OLE DB provider. The rowset name does not need to be specified in the external name. The query example passes in SQL statement text to retrieve the top three stores.

```
CREATE FUNCTION favorites (varchar(600))
  RETURNS TABLE (store_id CHAR (4),
                 name  VARCHAR (41),
                 sales INTEGER)
```

```
  SPECIFIC favorites
  LANGUAGE OLEDB
  EXTERNAL NAME '!!Provider=SQLOLEDB.1;Persist Security Info=False;
                 User ID=sa;Initial Catalog=pubs;Data Source=WALTZ;
                 Locale Identifier=1033;Use Procedure for Prepare=1;
                 Auto Translate=False;Packet Size=4096;Workstation ID=WALTZ;
                 OLE DB Services=CLIENTCURSOR;';

SELECT *
  FROM TABLE (favorites
             (' select top 3 sales.stor_id as store_id, ' CONCAT
                ' stores.stor_name as name, ' CONCAT
                ' sum(sales. qty) as sales ' CONCAT
              ' from sales, stores ' CONCAT
              ' where sales.stor_id = stores.stor_id ' CONCAT
              ' group by sales.stor_id, stores.stor_name ' CONCAT
              ' order by sum(sales.qty) desc ')) as f;
```

CREATE FUNCTION (sourced or template)

The CREATE FUNCTION (Sourced or Template) statement is used to register a function or function template with a server.

This statement can register the following objects:

- A user-defined function, based on another existing scalar or aggregate function, at the current server.
- A function template with an application server that is designated as a federated server. A *function template* is a partial function that contains no executable code. The user creates it for the purpose of mapping it to a data source function. After the mapping is created, the user can specify the function template in queries submitted to the federated server. When such a query is processed, the federated server will invoke the data source function to which the template is mapped, and return values whose data types correspond to those in the RETURNS portion of the template's definition.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

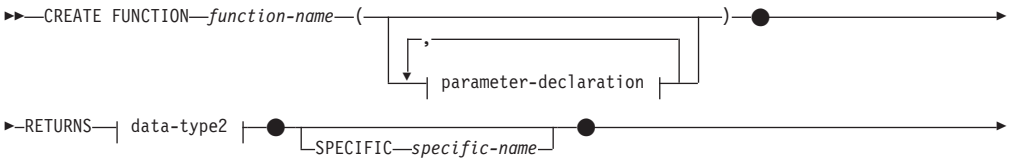
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the function does not exist
- CREATEIN privilege on the schema, if the schema name of the function exists
- DBADM authority

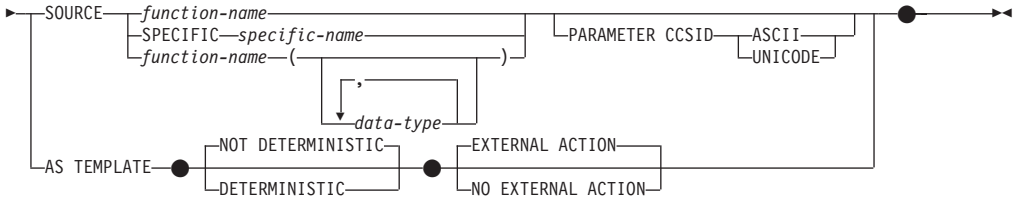
The privileges held by the authorization ID of the statement must also include EXECUTE privilege on the source function if the authorization ID of the statement does not have DATAACCESS authority and the SOURCE clause is specified.

Group privileges are not considered for any table or view specified in the CREATE FUNCTION statement.

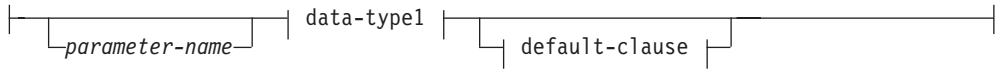
Syntax



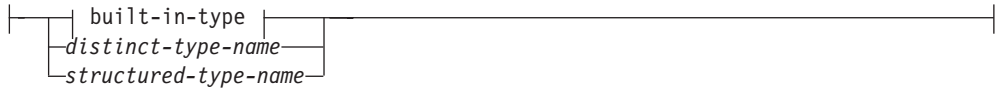
CREATE FUNCTION (sourced or template)



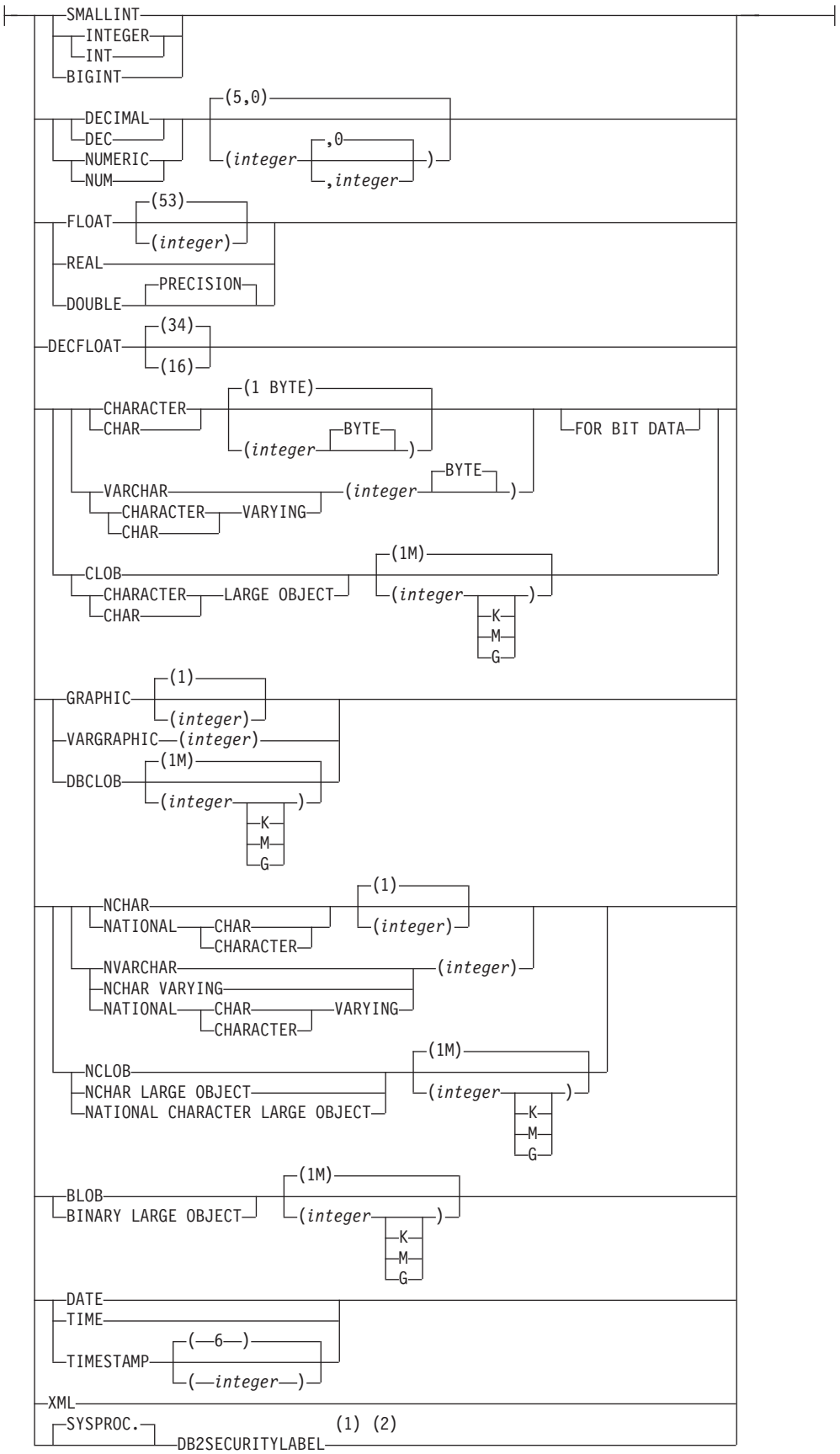
parameter-declaration:



data-type1, data-type2:



built-in-type:


SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer)
,0
,integer
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
DECFLOAT
(34)
(16)
CHARACTER
CHAR
(1 BYTE)
(integer)
BYTE
FOR BIT DATA
VARCHAR
CHARACTER
VARYING
CHAR
CLOB
LARGE OBJECT
(1M)
K
M
G
GRAPHIC
(1)
VARGRAPHIC
DBCLOB
NCHAR
NATIONAL
NVARCHAR
NCHAR VARYING
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
BLOB
BINARY LARGE OBJECT
DATE
TIME
TIMESTAMP
(6)
XML
SYSPROC.
DB2SECURITYLABEL
(1) (2)

default-clause:

Notes:

1 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

2 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.

Description

function-name
Names the function or function template being defined. It is a qualified or unqualified name that designates a function. The unqualified form of *function-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters and the data type of each parameter (without regard for any length, precision or scale attributes of the data type) must not identify a function or function template described in the catalog (SQLSTATE 42723). The unqualified name, together with the number and data types of the parameters, while of course unique within its schema, need not be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *function-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

When naming a user-defined function that is sourced on an existing function with the purpose of supporting the same function with a user-defined distinct type, the same name as the sourced function may be used. This allows users to use the same function with a user-defined distinct type without realizing that an additional definition was required. In general, the same name can be used for more than one function if there is some difference in the signature of the functions.

(*parameter-declaration***,...)**
Identifies the number of input parameters of the function or function template, and specifies the data type and optional default value of each parameter. One entry in the list must be specified for each parameter that the function or function template will expect to receive. No more than 90 parameters are allowed (SQLSTATE 54023).

It is possible to register a function that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example:

```
CREATE FUNCTION WOOFER() ...
```

No two identically-named functions within a schema are permitted to have exactly the same type for all corresponding parameters. This restriction also applies to a function and function template with the same name within the same schema. Lengths, precisions, and scales are not considered in this type comparison. Therefore, CHAR(8) and CHAR(35) are considered to be the same type, as are DECIMAL(11,2) and DECIMAL (4,3). A weakly typed distinct type specified for a parameter is considered to be the same data type as the source type of the distinct type. For a Unicode database, CHAR(13) and GRAPHIC(8) are considered to be the same type. There is some further bundling of types that causes them to be treated as the same type for this purpose, such as DECIMAL and NUMERIC. A duplicate signature returns an error (SQLSTATE 42723).

parameter-name
Specifies an optional name for the input parameter. The name cannot be the same as any other *parameter-name* in the parameter list (SQLSTATE 42734).

data-type1
Specifies the data type of the input parameter. The data type can be a built-in data type, a distinct type, or a structured type.

Any valid SQL data type can be used if it is castable to the type of the corresponding parameter of the function identified in the SOURCE clause (for information, see "Casting between data types"). However, this checking does not guarantee that an error will not occur when the function is invoked.

For a more complete description of each built-in data type, see "CREATE TABLE".

- A datetime type parameter is passed as a character data type, and the data is passed in the ISO format.
- Array types cannot be specified (SQLSTATE 42879).
- A reference type specified as REF(*type-name*) cannot be specified (SQLSTATE 42879).

For a user-defined distinct type, the length, precision, or scale attributes for the parameter are those of the source type of the distinct type (those specified on CREATE TYPE). A distinct type parameter is passed as the source type of the distinct type. If the name of the distinct type is unqualified, the database manager resolves the schema name by searching the schemas in the SQL path.

For a user-defined structured type, the appropriate transform functions must exist in the associated transform group.

Because the function is sourced, it is not necessary (but still permitted) to specify length, precision, or scale for the parameterized data types. Empty parentheses can be used instead; for example, CHAR(). A *parameterized data type* is any one of the data types that can be defined with a specific length, scale, or precision. The parameterized data types are the string data types, the decimal data types, and the TIMESTAMP data type.

With a function template, empty parentheses can also be used instead of specifying length, precision, or scale for the parameterized data types. It is recommended to use empty parentheses for the parameterized data types.

If you use empty parentheses, the length, precision, or scale is the same as that of the remote function, which is determined when the function template is mapped to a remote function by creating a function mapping. If you omit parentheses altogether, the default length for the data type is used (see "CREATE TABLE").

DEFAULT
Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression, or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see *default-clause* in the CREATE TABLE statement). Other special registers can be specified as the default by using an expression.

The expression can be any expression of the type described in "Expressions". If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified for a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB).

RETURNS
This mandatory clause identifies the output of the function or function template.

data-type2
Specifies the data type of the output.

With a sourced scalar function, any valid SQL data type is acceptable, as is a distinct type, provided it is castable from the result type of the source function. An array type cannot be specified as the data type of a parameter (SQLSTATE 42879).

The parameter of a parameterized type need not be specified for parameters of a sourced function. Instead, empty parentheses can be used; for example, VARCHAR().

For additional considerations and rules that apply to the specification of the data type in the RETURNS clause when the function is sourced on another, see the "Rules" section of this statement.

With a function template, empty parentheses are not allowed (SQLSTATE 42611). Length, precision, or scale must be specified for the parameterized data types. It is recommended to specify the same length, precision, or scale as that of the remote function.

SPECIFIC *specific-name*
Provides a unique name for the instance of the function that is being defined. This specific name can be used when sourcing on this function, dropping the function, or commenting on the function. It can never be used to invoke the function. The unqualified form of *specific-name* is an SQL identifier. The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another function instance that exists at the application server; otherwise an error (SQLSTATE 42710) is returned.

The *specific-name* may be the same as an existing *function-name*.

If no qualifier is specified, the qualifier that was used for *function-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *function-name* or an error (SQLSTATE 42882) is returned.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmmssxxx.

SOURCE

Specifies that the new function is being defined as a sourced function. A *sourced function* is implemented by another function (the *source function*). The function must be a scalar or aggregate function that exists at the current server, and it must be one of the following types of functions:

- A function that was defined with a CREATE FUNCTION statement
- A cast function that was generated by a CREATE TYPE statement
- A built-in function

If the source function is not a built-in function, the particular function can be identified by its name, function signature, or specific name.

If the source function is a built-in function, the SOURCE clause must include a function signature for the built-in function. The source function must not be any of the following built-in functions (If a particular syntax is indicated, only the indicated form cannot be specified.):

- CARDINALITY
- CHAR when more than one argument is specified and the first argument is a datetime data type
- CHARACTER_LENGTH
- COALESCE
- CONTAINS
- CURSOR_ROWCOUNT
- DATAPARTITIONNUM
- DBPARTITIONNUM
- DEREF
- EXTRACT
- GRAPHIC when more than one argument is specified and the first argument is a datetime data type
- GREATEST
- HASHEDVALUE
- INSERT when more than four arguments are specified
- INSTR when more than four arguments are specified
- LCASE when more than three arguments are specified
- LEAST
- LEFT when more than two arguments are specified
- LENGTH when more than one argument is specified
- LOCATE when more than three arguments are specified
- LOCATE_IN_STRING when more than four arguments are specified
- LOWER when more than three arguments are specified
- MAX
- MAX_CARDINALITY

- MIN
- NODENUMBER
- NULLIF
- NVL
- OVERLAY
- PARAMETER
- POSITION
- RAISE_ERROR
- REC2XML
- RID
- RID_BIT
- RIGHT when more than two arguments are specified
- SCORE
- STRIP
- SUBSTRING
- TRIM
- TRIM_ARRAY
- TYPE_ID
- TYPE_NAME
- TYPE_SCHEMA
- UCASE when more than three arguments are specified
- UPPER when more than three arguments are specified
- VALUE
- VARCHAR when more than one argument is specified and the first argument is a datetime data type
- VARGRAPHIC when more than one argument is specified and the first argument is a datetime data type
- XMLATTRIBUTES
- XMLCOMMENT
- XMLCONCAT
- XMLDOCUMENT
- XMLELEMENT
- XMLFOREST
- XMLNAMESPACES
- XMLPARSE
- XMLPI
- XMLQUERY
- XMLROW
- XMLSERIALIZE
- XMLTEXT
- XMLVALIDATE
- XMLXSROBJECTID
- XSLTRANSFORM

function-name
Identifies the particular function that is to be used as the source and is

valid only if there is exactly one specific function in the schema with this *function-name* for which the authorization ID of the statement has EXECUTE privilege. This syntax variant is not valid for a source function that is a built-in function.

If an unqualified name is provided, then the current SQL path (the value of the CURRENT PATH special register) is used to locate the function. The first schema in the SQL path that has a function with this name for which the authorization ID of the statement has EXECUTE privilege is selected.

If no function by this name exists in the named schema or if the name is not qualified and there is no function with this name in the SQL path, an error (SQLSTATE 42704) is returned. If there is more than one authorized specific instance of the function in the named or located schema, an error (SQLSTATE 42725) is returned. If a function by this name exists and the authorization ID of the statement does not have EXECUTE privilege on this function, an error (SQLSTATE 42501) is returned.

SPECIFIC *specific-name*
Identifies the particular user-defined function that is to be used as the source, by the *specific-name* either specified or defaulted to at function creation time. This syntax variant is not valid for a source function that is a built-in function.

If an unqualified name is provided, the current SQL path is used to locate the function. The first schema in the SQL path that has a function with this specific name for which the authorization ID of the statement has EXECUTE privilege is selected.

If no function by this *specific-name* exists in the named schema or if the name is not qualified and there is no function with this *specific-name* in the SQL path, an error (SQLSTATE 42704) is returned. If a function by this *specific-name* exists, and the authorization ID of the statement does not have EXECUTE privilege on this function, an error (SQLSTATE 42501) is returned.

function-name (data-type,...)
Provides the function signature, which uniquely identifies the source function. This is the only valid syntax variant for a source function that is a built-in function.

The rules for function resolution are applied to select one function from the functions with the same function name, given the data types specified in the SOURCE clause. However, the data type of each parameter in the function selected must have the exact same type as the corresponding data type specified in the source function.

function-name
Gives the function name of the source function. If an unqualified name is provided, then the schemas of the user's SQL path are considered.

data-type
Must match the data type that was specified on the CREATE FUNCTION statement in the corresponding position (comma separated).

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match. For example, DECIMAL() will match a parameter whose data type was defined as DECIMAL(7,2)).

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement. This can be useful in assuring that the intended function will be used. Note also that synonyms for data types will be considered a match (for example DEC and NUMERIC will match).

A type of FLOAT(n) does not need to match the defined value for n, because 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no function with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is returned.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the function. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031). When the function is invoked, the application code page for the function is the database code page.

UNICODE
Specifies that string data is encoded in Unicode. If the database is a Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8. In either case, when the function is invoked, the application code page for the function is 1208.

The PARAMETER CCSID clause must specify the same encoding scheme as the source function (SQLSTATE 53090).

AS TEMPLATE
Indicates that this statement will be used to create a function template, not a function with executable code.

NOT DETERMINISTIC or DETERMINISTIC
Specifies whether the function returns the same results for identical input arguments. The default is NOT DETERMINISTIC.

NOT DETERMINISTIC
Specifies that the function might not return the same result each time that the function is invoked with the same input arguments. The function depends on some state values that affect the results. The database manager uses this information during optimization of SQL statements. An example of a function that is not deterministic is one that generates random numbers.

A function that is not deterministic might receive incorrect results if it is executed by parallel tasks.

DETERMINISTIC
Specifies that the function always returns the same result each time that the function is invoked with the same input arguments. The database manager uses this information during optimization of SQL

statements. An example of a function that is deterministic is one that calculates the square root of the input argument.

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the function takes an action that changes the state of an object that the database manager does not manage. An example of an external action is sending a message or writing a record to a file. The default is EXTERNAL ACTION.

EXTERNAL ACTION
Specifies that the function takes an action that changes the state of an object that the database manager does not manage. EXTERNAL ACTION must be implicitly or explicitly specified if the SQL routine body invokes a function that is defined with EXTERNAL ACTION (SQLSTATE 428C2).

A function with external actions might return incorrect results if the function is executed by parallel tasks. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead of once for the function.

NO EXTERNAL ACTION
Specifies that the function does not take any action that changes the state of an object that the database manager does not manage. The database manager uses this information during optimization of SQL statements.

Rules

- For convenience, in this section the function being created will be called CF and the function identified in the SOURCE clause will be called SF, no matter which of the three allowable syntaxes was used to identify SF.
  - The unqualified name of CF and the unqualified name of SF can be different.
  - A function named as the source of another function can, itself, use another function as its source. Extreme care should be exercised when exploiting this facility, because it could be very difficult to debug an application if an indirectly invoked function returns an error.
  - The following clauses are invalid if specified in conjunction with the SOURCE clause (because CF will inherit these attributes from SF):
    - CAST FROM ...,
    - EXTERNAL ...,
    - LANGUAGE ...,
    - PARAMETER STYLE ...,
    - DETERMINISTIC / NOT DETERMINISTIC,
    - FENCED / NOT FENCED,
    - RETURNS NULL ON NULL INPUT / CALLED ON NULL INPUT
    - EXTERNAL ACTION / NO EXTERNAL ACTION
    - NO SQL / CONTAINS SQL / READS SQL DATA
    - SCRATCHPAD / NO SCRATCHPAD
    - FINAL CALL / NO FINAL CALL
    - RETURNS TABLE (...)
    - CARDINALITY ...
    - ALLOW PARALLEL / DISALLOW PARALLEL
    - DBINFO / NO DBINFO

- THREADSAFE / NOT THREADSAFE
- INHERIT SPECIAL REGISTERS

An error (SQLSTATE 42613) will result from violation of these rules.

- The number of input parameters in CF must be the same as those in SF; otherwise an error (SQLSTATE 42624) is returned.
- It is not necessary for CF to specify length, precision, or scale for a parameterized data type in the case of:
  - The function's input parameters,
  - Its RETURNS parameter

  Instead, empty parentheses may be specified as part of the data type (for example: VARCHAR()) in order to indicate that the length/precision/scale will be the same as those of the source function, or determined by the casting.

  However, if length, precision, or scale is specified then the value in CF is checked against the corresponding value in SF as outlined in the remaining rules for input parameters and returns value.
- The specification of the input parameters of CF are checked against those of SF. The data type of each parameter of CF must either be the same as or be *castable* to the data type of the corresponding parameter of SF. If any parameter is not the same type or castable, an error (SQLSTATE 42879) is returned.

  Note that this rule provides no guarantee against an error occurring when CF is used. An argument that matches the data type and length or precision attributes of a CF parameter may not be assignable if the corresponding SF parameter has a shorter length or less precision. In general, parameters of CF should not have length or precision attributes that are greater than the attributes of the corresponding SF parameters.
- The specifications for the RETURNS data type of CF are checked against that of SF. The final RETURNS data type of SF, after any casting, must either be the same as or castable to the RETURNS data type of CF. Otherwise an error (SQLSTATE 42866) is returned.

  Note that this rule provides no guarantee against an error occurring when CF is used. A result value that matches the data type and length or precision attributes of the SF RETURNS data type may not be assignable if the CF RETURNS data type has a shorter length or less precision. Caution should be used when choosing to specify the RETURNS data type of CF as having length or precision attributes that are less than the attributes of the SF RETURNS data type.

Notes

- Determining whether one data type is castable to another data type does not consider length or precision and scale for parameterized data types such as CHAR and DECIMAL. Therefore, errors may occur when using a function as a result of attempting to cast a value of the source data type to a value of the target data type. For example, VARCHAR is castable to DATE but if the source type is actually defined as VARCHAR(5), an error will occur when using the function.
- When choosing the data types for the parameters of a user-defined function, consider the rules for promotion that will affect its input values (see “Promotion of data types”). For example, a constant which may be used as an input value could have a built-in data type different from the one expected and, more significantly, may not be promoted to the data type expected. Based on the rules for promotion, it is generally recommended to use the following data types for parameters:
  - INTEGER instead of SMALLINT

  - DOUBLE instead of REAL
  - VARCHAR instead of CHAR
  - VARGRAPHIC instead of GRAPHIC
- Creating a function with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- For a federated server to recognize a data source function, the function must map to a counterpart at the federated database. If the database contains no counterpart, the user must create the counterpart and then the mapping.

  The counterpart can be a function (scalar or source) or a function template. If the user creates a function and the required mapping, then, each time a query that specifies the function is processed, DB2 (1) compares strategies for invoking it with strategies for invoking the data source function, and (2) invokes the function that is expected to require less overhead.

  If the user creates a function template and the mapping, then each time a query that specifies the template is processed, DB2 invokes the data source function that it maps to, provided that an access plan for invoking this function exists.
- ***Privileges***: The definer of a function always receives the EXECUTE privilege on the function, as well as the right to drop the function. The definer of the function is also given the WITH GRANT OPTION if any of the following conditions apply:
  - The source function is a built-in function.
  - The definer of the function has EXECUTE WITH GRANT OPTION on the source function.
  - The function is a template.
- ***EXTERNAL ACTION functions***: If an EXTERNAL ACTION function is invoked in other than the outermost select list, the results are unpredictable since the number of times the function is invoked will vary depending on the access plan used.
- ***Setting of the default value***: Parameters of a function that are defined with a default value are set to their default value when the functions is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the function is invoked.
- **Create function mapping to table or row functions:** A create function mapping to remote functions that returns a table or a row is not supported in a federated database.
- **Inheriting SECURED or NOT SECURED attributes from the source function:** The sourced user-defined function inherits the SECURED or NOT SECURED attribute from the source function in which only the topmost user-defined function is considered. If the topmost user-defined function is secure, any nested user-defined functions are considered secure. The database manager does not validate whether those nested user-defined functions are secure. If those nested functions can access sensitive data, the user with SECADM authority needs to ensure those functions are allowed to access those data and that a change control audit procedure has been established for all changes to those functions.

Examples

- *Example 1:* Some time after the creation of Pellow's original CENTRE external scalar function, another user wants to create a function based on it, except this function is intended to accept only integer arguments.

```
CREATE FUNCTION MYCENTRE (INTEGER, INTEGER)
  RETURNS FLOAT
  SOURCE PELLOW.CENTRE (INTEGER, FLOAT)
```

- *Example 2:* A distinct type, HATSIZE, has been created based on the built-in INTEGER data type. It would be useful to have an AVG function to compute the average hat size of different departments. This is easily done as follows:

  ```
  CREATE FUNCTION AVG (HATSIZE) RETURNS HATSIZE
    SOURCE SYSIBM.AVG (INTEGER)
  ```

  The creation of the distinct type has generated the required cast function, allowing the cast from HATSIZE to INTEGER for the argument and from INTEGER to HATSIZE for the result of the function.

- *Example 3:* In a federated system, a user wants to invoke an Oracle UDF that returns table statistics in the form of values with double-precision floating points. The federated server can recognize this function only if there is a mapping between the function and a federated database counterpart. But no such counterpart exists. The user decides to provide one in the form of a function template, and to assign this template to a schema called NOVA. The user uses the following code to register the template with the federated server.

  ```
  CREATE FUNCTION NOVA.STATS (DOUBLE, DOUBLE)
    RETURNS DOUBLE
    AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
  ```

- *Example 4:* In a federated system, a user wants to invoke an Oracle UDF that returns the dollar amounts that employees of a particular organization earn as bonuses. The federated server can recognize this function only if there is a mapping between the function and a federated database counterpart. No such counterpart exists; thus, the user creates one in the form of a function template. The user uses the following code to register this template with the federated server.

  ```
  CREATE FUNCTION BONUS ()
    RETURNS DECIMAL (8,2)
    AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
  ```

CREATE FUNCTION (SQL scalar, table, or row)

The CREATE FUNCTION (SQL scalar, table, or row) statement is used to define a user-defined SQL scalar, table, or row function.

A *scalar function* returns a single value each time it is invoked, and is generally valid wherever an SQL expression is valid. A *table function* can be used in a FROM clause and returns a table. A *row function* can be used as a transform function and returns a row.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the function does not exist
- CREATEIN privilege on the schema, if the schema name of the function refers to an existing schema
- DBADM authority

and at least one of the following authorities on each table, view, or nickname identified in any fullselect:

- CONTROL privilege on that table, view, or nickname
- SELECT privilege on that table, view, or nickname
- DATAACCESS authority

Group privileges other than PUBLIC are not considered for any table or view specified in the CREATE FUNCTION statement.

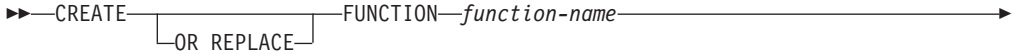
Authorization requirements of the data source for the table or view referenced by the nickname are applied when the function is invoked. The authorization ID of the connection can be mapped to a different remote authorization ID.

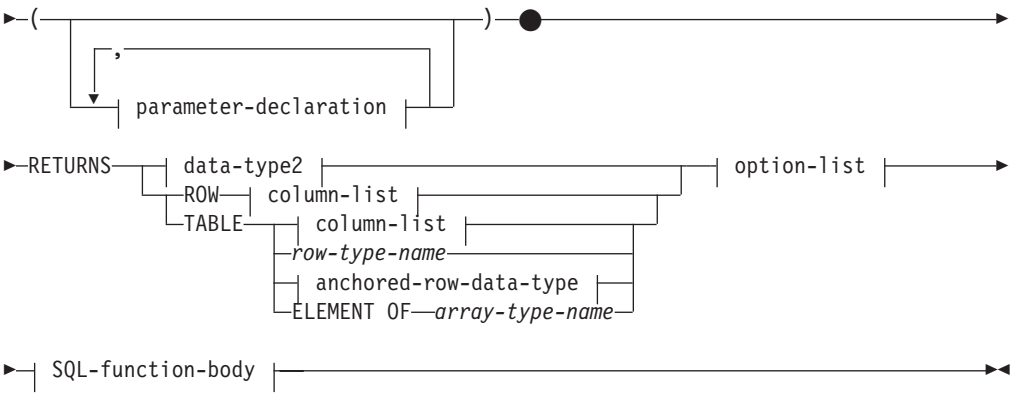
The privileges held by the authorization ID of the statement must also include all of the privileges necessary to invoke the SQL statements that are specified in the function body.

To replace an existing function, the authorization ID of the statement must be the owner of the existing function (SQLSTATE 42501).

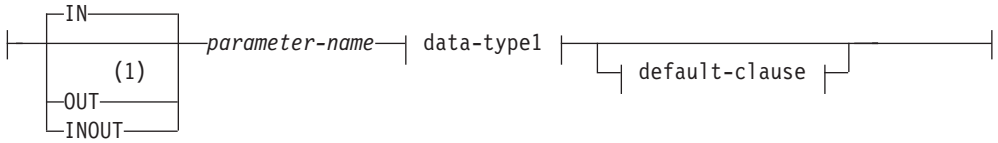
If the SECURED option is specified, the authorization ID of the statement must include SECADM or CREATE_SECURE_OBJECT authority (SQLSTATE 42501).

Syntax



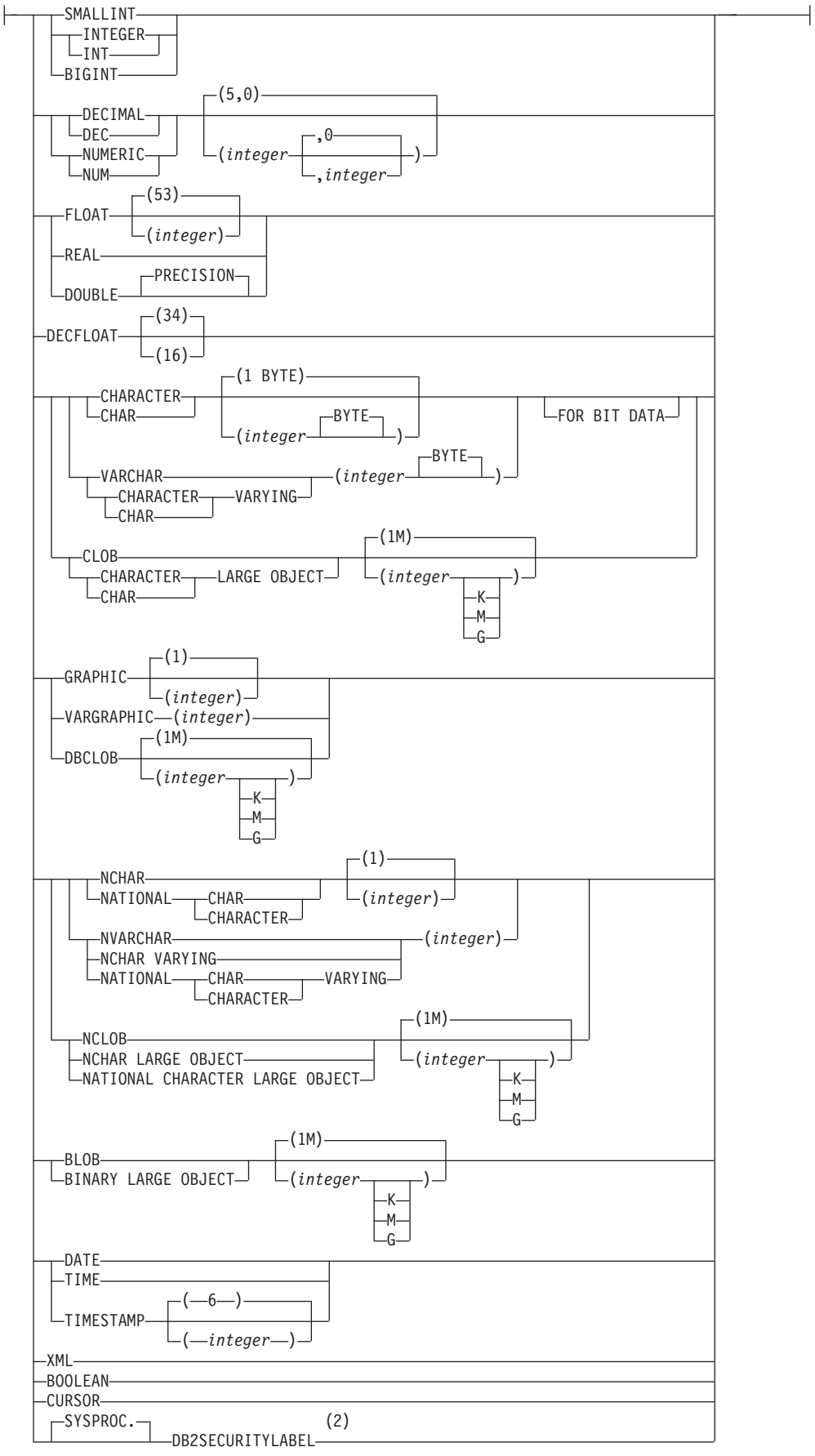


parameter-declaration:

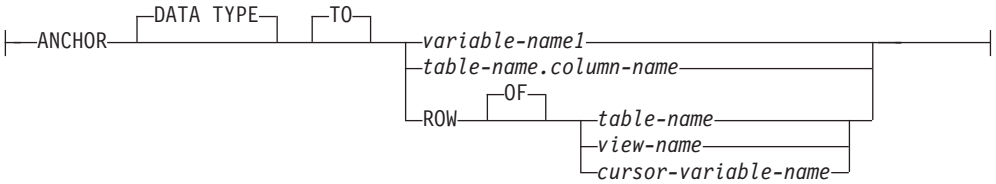


data-type1, data-type2:

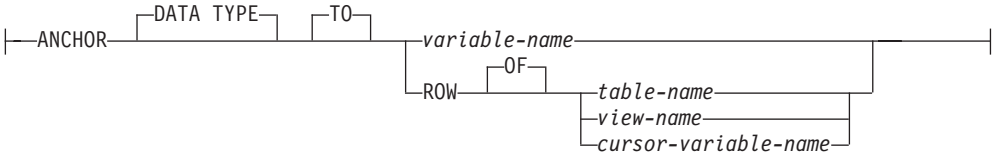
built-in-type:


SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer
,0
,integer)
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
DECFLOAT
(34)
(16)
CHARACTER
CHAR
(1 BYTE)
(integer
BYTE)
FOR BIT DATA
VARCHAR
CHARACTER
CHAR
VARYING
(integer
BYTE)
CLOB
CHARACTER
CHAR
LARGE OBJECT
(1M)
(integer
K
M
G)
GRAPHIC
(1)
(integer)
VARGRAPHIC
(integer)
DBCLOB
(1M)
(integer
K
M
G)
NCHAR
NATIONAL
CHAR
CHARACTER
(1)
(integer)
NVARCHAR
NCHAR VARYING
NATIONAL
CHAR
CHARACTER
VARYING
(integer)
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
(1M)
(integer
K
M
G)
BLOB
BINARY LARGE OBJECT
(1M)
(integer
K
M
G)
DATE
TIME
TIMESTAMP
(6)
(integer)
XML
BOOLEAN
CURSOR
SYSPROC.
DB2SECURITYLABEL
(2)

anchored-data-type:

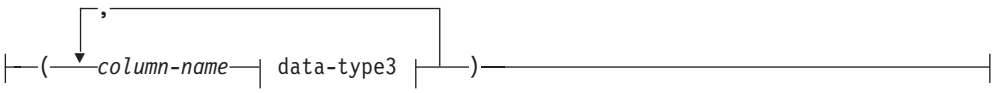


anchored-row-data-type:



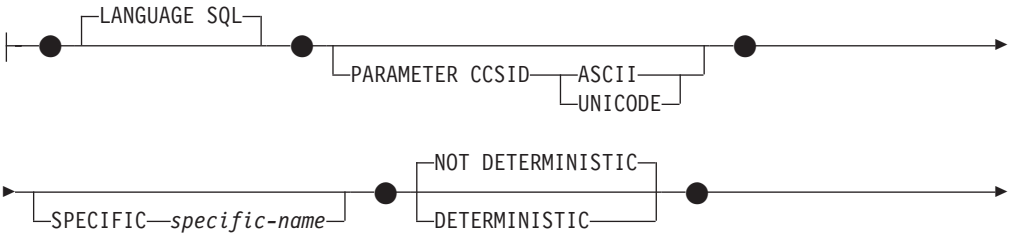
default-clause:

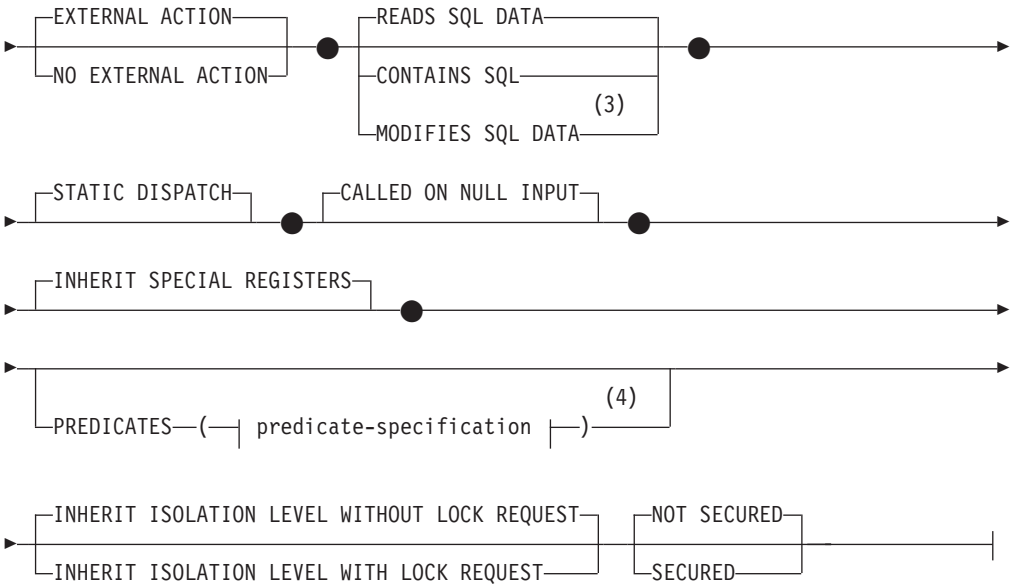
column-list:



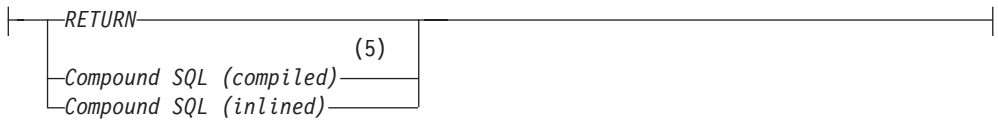
data-type3:

option-list:





SQL-function-body:



Notes:

1 OUT and INOUT are valid only if RETURNS specifies a scalar result and the SQL-function-body is a compound SQL (compiled) statement.

2 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

3 Valid only for compiled scalar function definition and an inlined table function definition. A compiled scalar function defined as MODIFIES SQL DATA can only be used as the only element on the right side of an assignment statement that is within a compound SQL (compiled) statement..

4 Valid only if RETURNS specifies a scalar result (*data-type2*)

5 The following apply to the specification of a compound SQL (compiled) statement: a) Must be used if the parameter data types or returned data types include a row type, array type, or cursor type; b) Must be used if the RETURNS TABLE clause specifies any syntax other than a column-list; c) Not supported if RETURNS ROW is specified; d) Not supported when defining a table function in a partitioned database environment.

Description

OR REPLACE

Specifies to replace the definition for the function if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the function are not affected. This option can be specified only by the owner of the object. This option is ignored if a definition for the function does not exist at the current server. To replace an existing function, the specific

name and function name of the new definition must be the same as the specific name and function name of the old definition, or the signature of the new definition must match the signature of the old definition. Otherwise, a new function is created.

If the function is referenced in the definition of a row permission or a column mask, the function cannot be replaced (SQLSTATE 42893).

function-name
Names the function being defined. It is a qualified or unqualified name that designates a function. The unqualified form of *function-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters and the data type of each parameter (without regard for any length, precision or scale attributes of the data type) must not identify a function described in the catalog (SQLSTATE 42723). The unqualified name, together with the number and data types of the parameters, while of course unique within its schema, need not be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *function-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

The same name can be used for more than one function if there is some difference in the signature of the functions. Although there is no prohibition against it, an external user-defined table function should not be given the same name as a built-in function.

(*parameter-declaration***, ...)**
Identifies the number of input parameters of the function, and specifies the mode, name, data type, and optional default value of each parameter. One entry in the list must be specified for each parameter that the function will expect to receive. No more than 90 parameters are allowed (SQLSTATE 54023).

It is possible to register a function that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example:

```
CREATE FUNCTION WOOFER() ...
```

No two identically-named functions within a schema are permitted to have exactly the same type for all corresponding parameters. Lengths, precisions, and scales are not considered in this type comparison. Therefore, CHAR(8) and CHAR(35) are considered to be the same type, as are DECIMAL(11,2) and DECIMAL (4,3), as well as DECFLOAT(16) and DECFLOAT(34). A weakly typed distinct type specified for a parameter is considered to be the same data type as the source type of the distinct type. For a Unicode database, CHAR(13) and GRAPHIC(8) are considered to be the same type. There is some further bundling of types that causes them to be treated as the same type for this purpose, such as DECIMAL and NUMERIC. A duplicate signature returns an error (SQLSTATE 42723).

If the data type for a parameter is a Boolean data type, array type, cursor type, or row type, the SQL function body can only reference the parameter within a compound SQL (compiled) statement (SQLSTATE 428H2).

IN | OUT | INOUT
Specifies the mode of the parameter. If an error is returned by the function, OUT parameters are undefined and INOUT parameters are unchanged. The default is IN.

IN Identifies the parameter as an input parameter to the function. Any changes made to the parameter within the function are not available to the invoking context when control is returned.

OUT
Identifies the parameter as an output parameter for the function.

The function must be a scalar function that is defined with a compound SQL (compiled) statement (SQLSTATE 42613).

The function can be referenced only on the right side of an assignment statement that is in a compound SQL (compiled) statement, and the function reference cannot be part of an expression (SQLSTATE 42887).

INOUT
Identifies the parameter as both an input and output parameter for the function.

The function must be a scalar function that is defined with a compound SQL (compiled) statement (SQLSTATE 42613).

The function can be referenced only on the right side of an assignment statement that is in a compound SQL (compiled) statement, and the function reference cannot be part of an expression (SQLSTATE 42887).

parameter-name
Specifies a name for the parameter. The name cannot be the same as any other *parameter-name* in the parameter list (SQLSTATE 42734).

data-type1
Specifies the data type of the parameter.

built-in-type
Specifies a built-in data type. For a more complete description of each built-in data type except BOOLEAN and CURSOR, which cannot be specified for a table, see "CREATE TABLE".

BOOLEAN
For a Boolean.

CURSOR
For a reference to an underlying cursor.

anchored-data-type
Identifies another object used to define the parameter data type. The data type of the anchor object can be any of the data types explicitly allowed as *data-type1*. The data type of the anchor object has the same limitations that apply to specifying the data type directly, or in the case of a row, to creating a row type.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name1
Identifies a global variable. The data type of the global variable is used as the data type for *parameter-name*.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for *parameter-name*.

ROW OF *table-name* **or** *view-name*
Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data type of *parameter-name* is an unnamed row type.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following elements (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a *select-statement* where all the result columns are named.

If the cursor type of the cursor variable is not strongly typed using a named row type, the data type of *parameter-name* is an unnamed row type.

array-type-name
Specifies the name of a user-defined array type. If *array-type-name* is specified without a schema name, the array type is resolved by searching the schemas in the SQL path.

cursor-type-name
Specifies the name of a cursor type. If *cursor-type-name* is specified without a schema name, the cursor type is resolved by searching the schemas in the SQL path.

distinct-type-name
Specifies the name of a distinct type. The length, precision, and scale of the parameter are, respectively, the length, precision, and scale of the source type of the distinct type. A distinct type parameter is passed as the source type of the distinct type. If *distinct-type-name* is specified without a schema name, the distinct type is resolved by searching the schemas in the SQL path.

REF (*type-name***)**
Specifies a reference type without a scope. The specified *type-name* must identify a user-defined structured type (SQLSTATE 428DP). The system does not attempt to infer the scope of the parameter or result. Inside the body of the function, a reference type can be used in a dereference operation only by first casting it to have a scope. Similarly, a reference returned by an SQL function can be used in a dereference operation only by first casting it to have a scope. If a type name is specified without a schema name, the *type-name* is resolved by searching the schemas in the SQL path.

row-type-name
Specifies the name of a user-defined row type. The fields of the parameter are the fields of the row type. If *row-type-name* is specified without a schema name, the row type is resolved by searching the schemas in the SQL path.

structured-type-name
Specifies the name of a user-defined structured type. If *structured-type-name* is specified without a schema name, the structured type is resolved by searching the schemas in the SQL path.

DEFAULT
Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression, or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see *default-clause* in the CREATE TABLE statement). Other special registers can be specified as the default by using an expression.

The expression can be any expression of the type described in "Expressions". If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified in the following situations:

- For INOUT or OUT parameters (SQLSTATE 42601)
- For a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB)
- For a parameter to a function definition that also specified RETURNS ROW or a PREDICATES clause (SQLSTATE 42613)

RETURNS
This mandatory clause identifies the type of output of the function.

If the data type of the output of the function is a Boolean data type, array type, cursor type, or row type, the SQL function body must be a compound SQL (compiled) statement (SQLSTATE 428H2).

data-type2
Specifies the data type of the output.

In this statement, exactly the same considerations apply as for the parameters of SQL functions described previously in *data-type1* for function parameters.

ROW
Specifies that the output of the function is a single row. If the function returns more than one row, an error is returned (SQLSTATE 21505).

This form of a row function can be used only as a transform function for a structured type (having one structured type as its parameter and returning only built-in data types).

column-list
The list of column names and data types returned for a ROW function. The *column-list* must include at least two columns (SQLSTATE 428F0).

column-name
Specifies the name of this column. The name cannot be qualified and the same name cannot be used for more than one column in the list.

data-type3
Specifies the data type of the column, and can be any data type supported by a parameter of the SQL function.

The same considerations apply as for the parameters of SQL functions described previously in *data-type1* for function parameters. However, *data-type3* does not support *anchored-data-type*, *array-type-name*, *cursor-type-name*, and *row-type-name*.

TABLE
Specifies that the output of the function is a table.

column-list
The list of column names and data types returned for a TABLE function

column-name
Specifies the name of this column. The name cannot be qualified and the same name cannot be used for more than one column in the list.

data-type3
Specifies the data type of the column, and can be any data type supported by a parameter of the SQL function.

The same considerations apply as for the parameters of SQL functions described previously in *data-type1* for function parameters. However, *data-type3* does not support *anchored-data-type*, *array-type-name*, *cursor-type-name*, and *row-type-name*.

row-type-name
Specifies a row type from which the fields are used to derive the column list. The field names of the row type are used as the column names.

anchored-row-data-type
Identifies row information from another object to use as the columns of the returned table.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies a global variable. The data type of the referenced variable must be a row type.

ROW OF *table-name* **or** *view-name*
Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data types of the anchor object columns have the same limitations that apply to *data-type3*.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based

on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following objects (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type.
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a select-statement where all the result columns are named.

ELEMENT OF *array-type-name*
Specifies an array type from which the element data type is used to derive the column list. If *array-type-name* identifies an array type with elements that are a row type, the field names of the row type are used as the column names. If the *array-type-name* identifies an array type with elements that are not row types, the single result column name is COLUMN_VALUE.

SPECIFIC *specific-name*
Provides a unique name for the instance of the function that is being defined. This specific name can be used when sourcing on this function, dropping the function, or commenting on the function. It can never be used to invoke the function. The unqualified form of *specific-name* is an SQL identifier. The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another function instance that exists at the application server; otherwise an error is raised (SQLSTATE 42710).

The *specific-name* may be the same as an existing *function-name*.

If no qualifier is specified, the qualifier that was used for *function-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *function-name* or an error is raised (SQLSTATE 42882).

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmmssxxx.

LANGUAGE SQL
Specifies that the function is written using SQL.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the function. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE
Specifies that character data is in UTF-8, and that graphic data is in UCS-2. If the database is not a Unicode database, PARAMETER CCSID UNICODE cannot be specified (SQLSTATE 56031).

DETERMINISTIC or NOT DETERMINISTIC
This optional clause specifies whether the function always returns the same results for given argument values (DETERMINISTIC) or whether the function depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC function must always return the same table from

successive invocations with identical inputs. Optimizations taking advantage of the fact that identical inputs always produce the same results are prevented by specifying NOT DETERMINISTIC.

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the function takes an action that changes the state of an object that the database manager does not manage. An example of an external action is sending a message or writing a record to a file. The default is EXTERNAL ACTION.

EXTERNAL ACTION
Specifies that the function takes an action that changes the state of an object that the database manager does not manage.

NO EXTERNAL ACTION
Specifies that the function does not take any action that changes the state of an object that the database manager does not manage. The database manager uses this information during optimization of SQL statements.

CONTAINS SQL, READS SQL DATA, or MODIFIES SQL DATA
Indicates what type of SQL statements can be executed.

CONTAINS SQL
Indicates that SQL statements that neither read nor modify SQL data can be executed by the function (SQLSTATE 42985).

READS SQL DATA
Indicates that SQL statements that do not modify SQL data can be executed by the function (SQLSTATE 42985).

MODIFIES SQL DATA
Indicates that all SQL statements supported in the SQL-function-body can be executed by the function.

STATIC DISPATCH
This optional clause indicates that at function resolution time, DB2 chooses a function based on the static types (declared types) of the parameters of the function.

CALLED ON NULL INPUT
This clause indicates that the function is called regardless of whether any of its arguments are null. It can return a null value or a non-null value. Responsibility for testing null argument values lies with the user-defined function.

The phrase NULL CALL may be used in place of CALLED ON NULL INPUT.

INHERIT SPECIAL REGISTERS
This optional clause indicates that updatable special registers in the function will inherit their initial values from the environment of the invoking statement. For a function that is invoked in the select-statement of a cursor, the initial values are inherited from the environment when the cursor is opened. For a routine that is invoked in a nested object (for example, a trigger or a view), the initial values are inherited from the runtime environment (not the object definition).

No changes to the special registers are passed back to the caller of the function.

Some special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore never inherited from the caller.

PREDICATES
For predicates using this function, this clause identifies those that can exploit the index extensions, and can use the optional SELECTIVITY clause for the predicate's search condition. If the PREDICATES clause is specified, the function must be defined as DETERMINISTIC with NO EXTERNAL ACTION (SQLSTATE 42613). If the PREDICATES clause is specified, and the database is not a Unicode database, PARAMETER CCSID UNICODE must not be specified (SQLSTATE 42613). PREDICATES cannot be specified if SQL-function-body is a compound SQL (compiled) statement (SQLSTATE 42613).

predicate-specification
For details on predicate specification, see "CREATE FUNCTION (External Scalar)".

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST or INHERIT ISOLATION LEVEL WITH LOCK REQUEST
Specifies whether or not a lock request can be associated with the isolation-clause of the statement when the function inherits the isolation level of the statement that invokes the function. The default is INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST.

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
Specifies that, as the function inherits the isolation level of the invoking statement, it cannot be invoked in the context of an SQL statement which includes a lock-request-clause as part of a specified isolation-clause (SQLSTATE 42601).

INHERIT ISOLATION LEVEL WITH LOCK REQUEST
Specifies that, as the function inherits the isolation level of the invoking statement, it also inherits the specified lock-request-clause.

SQL-function-body
Specifies the body of the function. Parameter names can be referenced in the SQL-function-body. Parameter names may be qualified with the function name to avoid ambiguous references.

For RETURN statement, see: RETURN statement.

For *Compound SQL (compiled)*, see: Compound SQL (compiled) statement.

For *Compound SQL (inlined)*, see: Compound SQL (inlined) statement.

NOT SECURED or SECURED
Specifies whether the function is considered secure for row and column access control. The default is NOT SECURED.

NOT SECURED
Indicates that the function is not considered secure. When the function is invoked, the arguments of the function must not reference a column for which a column mask is enabled and column level access control is activated for its table (SQLSTATE 428HA). This rule applies to the non secure user-defined functions that are invoked anywhere in the statement.

SECURED
Indicates that the function is considered secure. The function must be secure when it is referenced in a row permission or a column mask (SQLSTATE 428H8).

Rules

- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row

definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

- ***Use of cursor and row types:*** A function that uses a cursor type or row type for a parameter or returns a cursor type or row type can only be invoked from within a compound SQL (compiled) statement (SQLSTATE 428H2).
- ***Table access restrictions***: If a function is defined as READS SQL DATA, no statement in the function can access a table that is being modified by the statement that invoked the function (SQLSTATE 57053). For example, suppose the user-defined function BONUS() is defined as READS SQL DATA. If the statement UPDATE EMPLOYEE SET SALARY = SALARY + BONUS(EMPNO) is invoked, no SQL statement in the BONUS function can read from the EMPLOYEE table.

  If a function defined with MODIFIES SQL DATA contains nested CALL statements, read access to the tables being modified by the function (by either the function definition or the statement that invoked the function) is not allowed (SQLSTATE 57053).
- ***Use in a partitioned database environment:*** In a partitioned database environment, a scalar function defined using a compound SQL (compiled) statement can be referenced only on the right side of an assignment statement and the function reference cannot be part of an expression. Such an assignment statement cannot be in a Compound SQL (inlined) statement.

Notes

- Resolution of function calls inside the function body is done according to the SQL path that is effective for the CREATE FUNCTION statement and does not change after the function is created.
- If an SQL function contains multiple references to any of the date or time special registers, all references return the same value, and it will be the same value returned by the register invocation in the statement that called the function.
- The body of an SQL function cannot contain a recursive call to itself or to another function or method that calls it, since such a function could not exist to be called.
- If an object referenced in the SQL function body does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the SQL function will still be created successfully. The SQL function will be marked invalid and will be revalidated the next time it is invoked.
- The following rules are enforced by all statements that create functions or methods:
  - A function may not have the same signature as a method (comparing the first *parameter-type* of the function with the *subject-type* of the method).
  - A function and a method may not be in an overriding relationship. That is, if the function were a method with its first parameter as subject, it must not override, or be overridden by, another method. For more information about overriding methods, see the “CREATE TYPE (Structured)” statement.
  - Because overriding does not apply to functions, it is permissible for two functions to exist such that, if they were methods, one would override the other.

  For the purpose of comparing parameter-types in the preceding rules:
  - Parameter-names, lengths, AS LOCATOR, and FOR BIT DATA are ignored.
  - A subtype is considered to be different from its supertype.

- ***Privileges***: The definer of a function always receives the EXECUTE privilege on the function, as well as the right to drop the function. The definer of a function is also given the WITH GRANT OPTION on the function if the definer has WITH GRANT OPTION on all privileges required to define the function, or if the definer has SYSADM or DBADM authority.

  The definer of a function only acquires privileges if the privileges from which they are derived exist at the time the function is created. The definer must have these privileges either directly, or because PUBLIC has the privileges. Privileges held by groups of which the function definer is a member are not considered. When using the function, the connected user's authorization ID must have the valid privileges on the table or view that the nickname references at the data source.
- ***Setting of the default value***: Parameters of a function that are defined with a default value are set to their default value when the functions is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the function is invoked.
- ***EXTERNAL ACTION functions***: If an EXTERNAL ACTION function is invoked in other than the outermost select list, the results are unpredictable since the number of times the function is invoked will vary depending on the access plan used.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used:
  - NULL CALL can be specified in place of CALLED ON NULL INPUT

  The following syntax is accepted as the default behavior:
  - CCSID UNICODE in a Unicode database
  - CCSID ASCII in a non-Unicode database
- ***Creating a secure function***: Normally users with SECADM authority do not have privileges to create database objects such as triggers or functions. Typically they will examine the data accessed by the function, ensure it is secure, then grant the CREATE_SECURE_OBJECT authority to someone who currently has required privileges to create a secure user-defined function. After the function is created, they will revoke the CREATE_SECURE_OBJECT authority from the function owner.

  The SECURED attribute is considered to be an assertion that declares the user has established a change control audit procedure for all changes to the user-defined function. The database manager assumes that such a control audit procedure is in place for all subsequent ALTER FUNCTION statements or changes to external packages.
- ***Invoking other user-defined functions in a secure function***: If a secure user-defined function invokes other user-defined functions, the database manager does not validate whether those nested user-defined functions have the SECURED attribute. If those nested functions can access sensitive data, the user with SECADM authority needs to ensure those functions are allowed to access those data and a change control audit procedure has been established for all changes to those functions.
- ***Replacing an existing function such that the secure attribute is changed (from SECURED to NOT SECURED and vice versa)***: Packages and dynamically cached SQL statements that depend on the function may be invalidated because the secure attribute affects the access path selection for statements involving tables for which row or column level access control is activated.

Examples

- *Example 1:* Define a scalar function that returns the tangent of a value using the existing sine and cosine functions.

```
CREATE FUNCTION TAN (X DOUBLE)
  RETURNS DOUBLE
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN SIN(X)/COS(X)
```

- *Example 2:* Define a transform function for the structured type PERSON.

```
CREATE FUNCTION FROMPERSON (P PERSON)
  RETURNS ROW (NAME VARCHAR(10), FIRSTNAME VARCHAR(10))
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN VALUES (P..NAME, P..FIRSTNAME)
```

- *Example 3:* Define a table function that returns the employees in a specified department number.

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
  RETURNS TABLE (EMPNO CHAR(6),
                 LASTNAME VARCHAR(15),
                 FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  READS SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN
    SELECT EMPNO, LASTNAME, FIRSTNME
      FROM EMPLOYEE
      WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
```

- *Example 4:* Define the table function from Example 3 with auditing.

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
  RETURNS TABLE (EMPNO CHAR(6),
                 LASTNAME VARCHAR(15),
                 FIRSTNAME VARCHAR(12))
  LANGUAGE SQL
  MODIFIES SQL DATA
  NO EXTERNAL ACTION
  DETERMINISTIC
  BEGIN ATOMIC
    INSERT INTO AUDIT
    VALUES (USER,
            'Table: EMPLOYEE Prd: DEPTNO = ' CONCAT DEPTNO);
    RETURN
      SELECT EMPNO, LASTNAME, FIRSTNME
        FROM EMPLOYEE
        WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
  END
```

- *Example 5:* Define a scalar function that reverses a string.

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(4000))
  RETURNS VARCHAR(4000)
  DETERMINISTIC NO EXTERNAL ACTION CONTAINS SQL
  BEGIN ATOMIC
  DECLARE REVSTR, RESTSTR VARCHAR(4000) DEFAULT '';
  DECLARE LEN INT;
  IF INSTR IS NULL THEN
  RETURN NULL;
  END IF;
```

```
  SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR));
  WHILE LEN > 0 DO
  SET (REVSTR, RESTSTR, LEN)
    = (SUBSTR(RESTSTR, 1, 1) CONCAT REVSTR,
    SUBSTR(RESTSTR, 2, LEN - 1),
    LEN - 1);
  END WHILE;
  RETURN REVSTR;
END
```

- *Example 6:* Create a function that increments a variable passed as an INOUT parameter and return any error as the return code.

```
CREATE FUNCTION increment(INOUT result INTEGER, IN delta INTEGER)
  RETURNS INTEGER
  BEGIN
    DECLARE code INTEGER DEFAULT 0;
    DECLARE SQLCODE INTEGER;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN
    SET code = SQLCODE;
    RETURN code;
  END;
  SET result = result + delta;
  RETURN code;
END@
```

- *Example 7:* Create a compiled SQL function that takes an XML document as input and returns the customer name.

```
CREATE FUNCTION get_customer_name_compiled(doc XML)
  RETURNS VARCHAR(25)
  BEGIN
    RETURN XMLCAST(XMLQUERY
       ('$d/customerinfo/name' PASSING doc AS "d")AS VARCHAR(25));
  END
```

- *Example 8:* Create a compiled SQL function that takes a phone number and a region number passed as IN parameters and returns the complete number in an OUT XML parameter.

```
CREATE FUNCTION construct_xml_phone
  (IN  phoneNo VARCHAR(20),
   IN  regionNo VARCHAR(8),
   OUT full_phone_xml XML)
  RETURNS VARCHAR(28)
  LANGUAGE SQL
  NO EXTERNAL ACTION
  BEGIN
    SET full_phone_xml = XMLELEMENT (NAME "phone", regionNo || phoneNo);
    RETURN regionNo || phoneNo;
  END
```

CREATE FUNCTION MAPPING

The CREATE FUNCTION MAPPING statement can define a mapping between a federated database function or function template and a data source function, or disable a default mapping between a federated database function and a data source function.

When defining a mapping, the CREATE FUNCTION MAPPING statement can associate the federated database function or template with a function at the following sources:

- A specified data source
- A range of data sources; for example, all data sources of a particular type and version

If multiple function mappings are applicable to a function, the most recent one is applied.

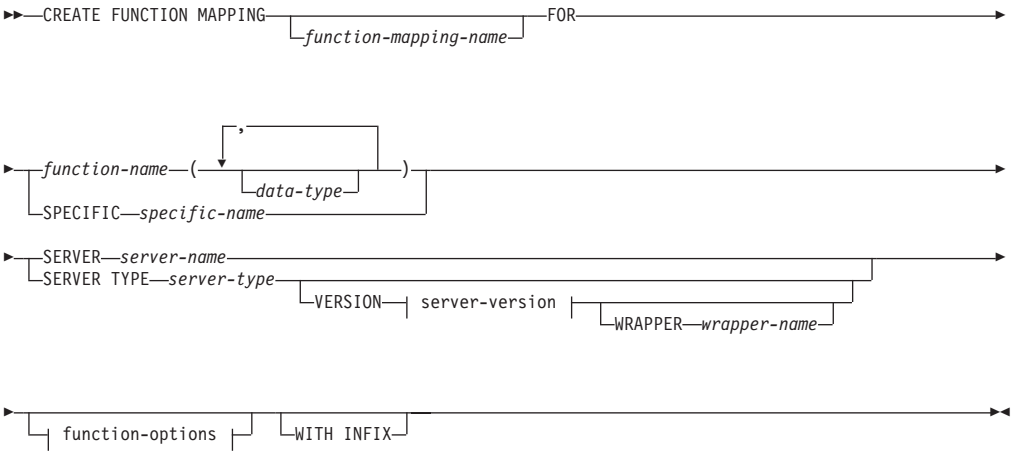
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

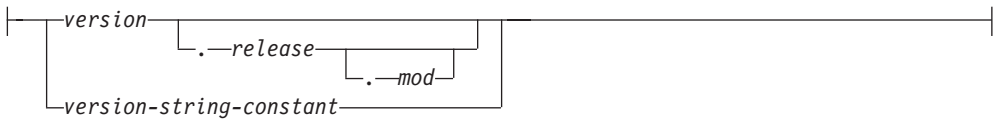
Authorization

The privileges held by the authorization ID of the statement must include DBADM authority.

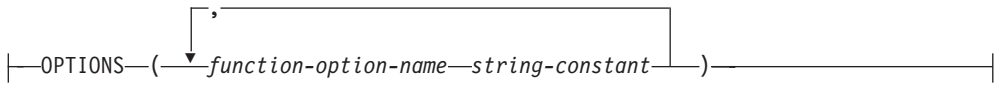
Syntax



server-version:



function-options:



Description

function-mapping-name
Names the function mapping. The name must not identify a function mapping that is already described in the catalog (SQLSTATE 42710).

If the *function-mapping-name* is omitted, a system-generated unique name is assigned.

function-name
Specifies the qualified or unqualified name of the federated database function or federated database function template from which to map.

data-type
For a function or function template that has input parameters, *data-type* specifies the data type of each parameter. The *data type* cannot be an XML or a user-defined type.

Empty parentheses can be used instead of specifying length, precision, or scale for the parameterized data types. It is recommended to use empty parentheses for the parameterized data types; for example, CHAR(). A parameterized data type is any one of the data types that can be defined with a specific length, scale, or precision. The parameterized data types are the string data types and the decimal data types. If you specify length, precision, or scale, it must be the same as that of the function template. If you omit parentheses altogether, the default length for the data type is used (see the description of the CREATE TABLE statement).

SPECIFIC *specific-name*
Identifies the function or function template from which to map. Specify *specific-name* to create a convenient function name.

SERVER *server-name*
Names the data source containing the function that is being mapped.

SERVER TYPE *server-type*
Identifies the type of data source containing the function that is being mapped.

VERSION
Identifies the version of the data source denoted by *server-type*.

version
Specifies the version number. The value must be an integer.

release
Specifies the number of the release of the version denoted by *version*. The value must be an integer.

mod
Specifies the number of the modification of the release denoted by *release*. The value must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release* and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*
Specifies the name of the wrapper that the federated server uses to interact with data sources of the type and version denoted by *server-type* and *server-version*.

OPTIONS
Indicates what function mapping options are to be enabled.

function-option-name
Names a function mapping option that applies either to the function mapping or to the data source function included in the mapping.

string-constant
Specifies the setting for *function-option-name* as a character string constant.

WITH INFIX
Specifies that the data source function be generated in infix format. The federated database system converts prefix notation to the infix notation that is used by the remote data source.

Notes

- A federated database function or function template can map to a data source function if:
  - The federated database function or template has the same number of input parameters as the data source function.
  - The data types that are defined for the federated function or template are compatible with the corresponding data types defined for the data source function.
- If a distributed request references a DB2 function that maps to a data source function, the optimizer develops strategies for invoking either function when the request is processed. The DB2 function is invoked if doing so requires less overhead than invoking the data source function. Otherwise, if invoking the DB2 function requires more overhead, the data source function is invoked.
- If a distributed request references a DB2 function template that maps to a data source function, only the data source function can be invoked when the request is processed. The template cannot be invoked because it has no executable code.
- Default function mappings can be rendered inoperable by disabling them (they cannot be dropped). To disable a default function mapping, code the CREATE FUNCTION MAPPING statement so that it specifies the name of the DB2 function within the mapping and sets the DISABLE option to 'Y'.
- Functions in the SYSIBM schema do not have a specific name. To override the default function mapping for a function in the SYSIBM schema, specify *function-name* using the explicit qualifier SYSIBM; for example, SYSIBM.LENGTH().
- A CREATE FUNCTION MAPPING statement within a given unit of work (UOW) cannot be processed (SQLSTATE 55007) under either of the following conditions:
  - The statement references a single data source, and the UOW already includes one of the following:
    - A SELECT statement that references a nickname for a table or view within this data source
    - An open cursor on a nickname for a table or view within this data source
    - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within this data source

  - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following:
    - A SELECT statement that references a nickname for a table or view within one of these data sources
    - An open cursor on a nickname for a table or view within one of these data sources
    - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within one of these data sources
- **Create function mapping to table or row functions:** A create function mapping to remote functions that returns a table or a row is not supported in a federated database.
- **Syntax alternatives:** The following syntax is supported for compatibility with previous versions of DB2:
  - ADD can be specified before *function-option-name string-constant*.

Examples

- *Example 1:* Map a function template to a UDF that all Oracle data sources can access. The template is called STATS and belongs to a schema called NOVA. The Oracle UDF is called STATISTICS and belongs to a schema called STAR.

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN1
  FOR NOVA.STATS (DOUBLE, DOUBLE)
  SERVER TYPE ORACLE
  OPTIONS (REMOTE_NAME 'STAR.STATISTICS')
```

- *Example 2:* Map a function template called BONUS to a UDF, also called BONUS, that is used at an Oracle data source called ORACLE1.

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN2
  FOR BONUS()
  SERVER ORACLE1
  OPTIONS (REMOTE_NAME 'BONUS')
```

- *Example 3:* Assume that there is a default function mapping between the WEEK system function that is defined to the federated database and a similar function that is defined to Oracle data sources. When a query that requests Oracle data and that references WEEK is processed, either WEEK or its Oracle counterpart will be invoked, depending on which one is estimated by the optimizer to require less overhead. The DBA wants to find out how performance would be affected if only WEEK were invoked for such queries. To ensure that WEEK is invoked each time, the DBA must disable the mapping.

```
CREATE FUNCTION MAPPING
  FOR SYSFUN.WEEK(INT)
  SERVER TYPE ORACLE
  OPTIONS (DISABLE 'Y')
```

- *Example 4:* Map the federated function UCASE(CHAR) to a UDF that is used at an Oracle data source called ORACLE2. Include the estimated number of instructions per invocation of the Oracle UDF.

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN4
  FOR SYSFUN.UCASE(CHAR)
  SERVER ORACLE2
  OPTIONS
    (REMOTE_NAME 'UPPERCASE',
    INSTS_PER_INVOC '1000')
```

CREATE GLOBAL TEMPORARY TABLE

The CREATE GLOBAL TEMPORARY TABLE statement creates a description of a temporary table at the current server. Each session that selects from a created temporary table retrieves only rows that the same session has inserted. When the session terminates, the rows of the table associated with the session are deleted.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

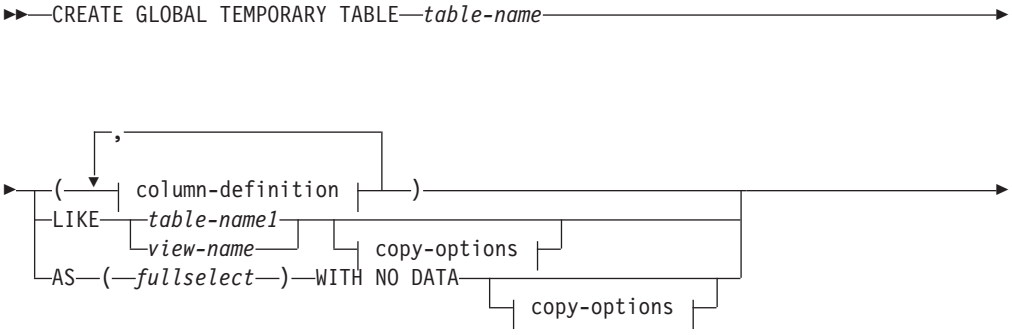
The privileges held by the authorization ID of the statement must include either DBADM authority, or CREATETAB authority in combination with further authorization, as described here:

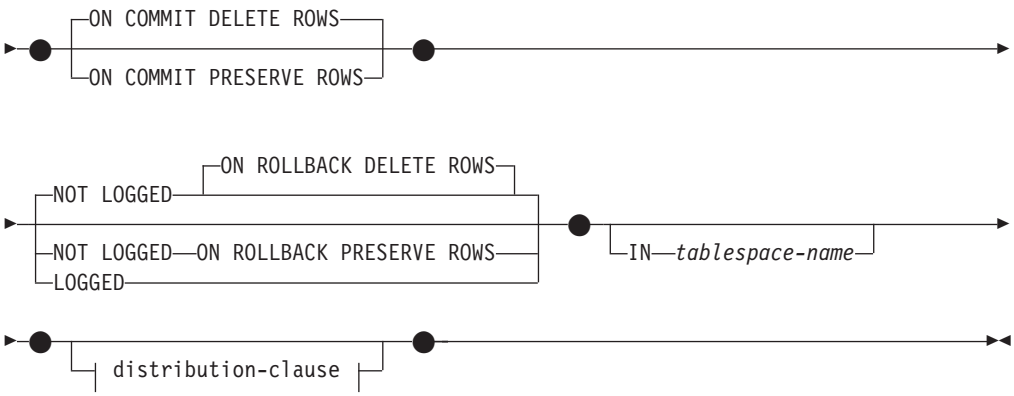
- One of the following privileges and authorities:
  - USE privilege on the table space
  - SYSADM
  - SYSCTRL
- Plus one of these privileges and authorities:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema

When defining a table using LIKE or a fullselect, the privileges held by the authorization ID of the statement must also include at least one of the following on each identified table or view:

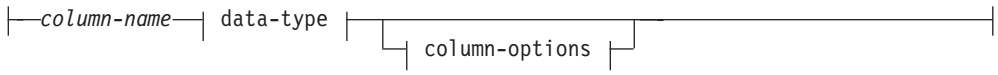
- SELECT privilege on the table or view
- CONTROL privilege on the table or view
- DATAACCESS authority

Syntax



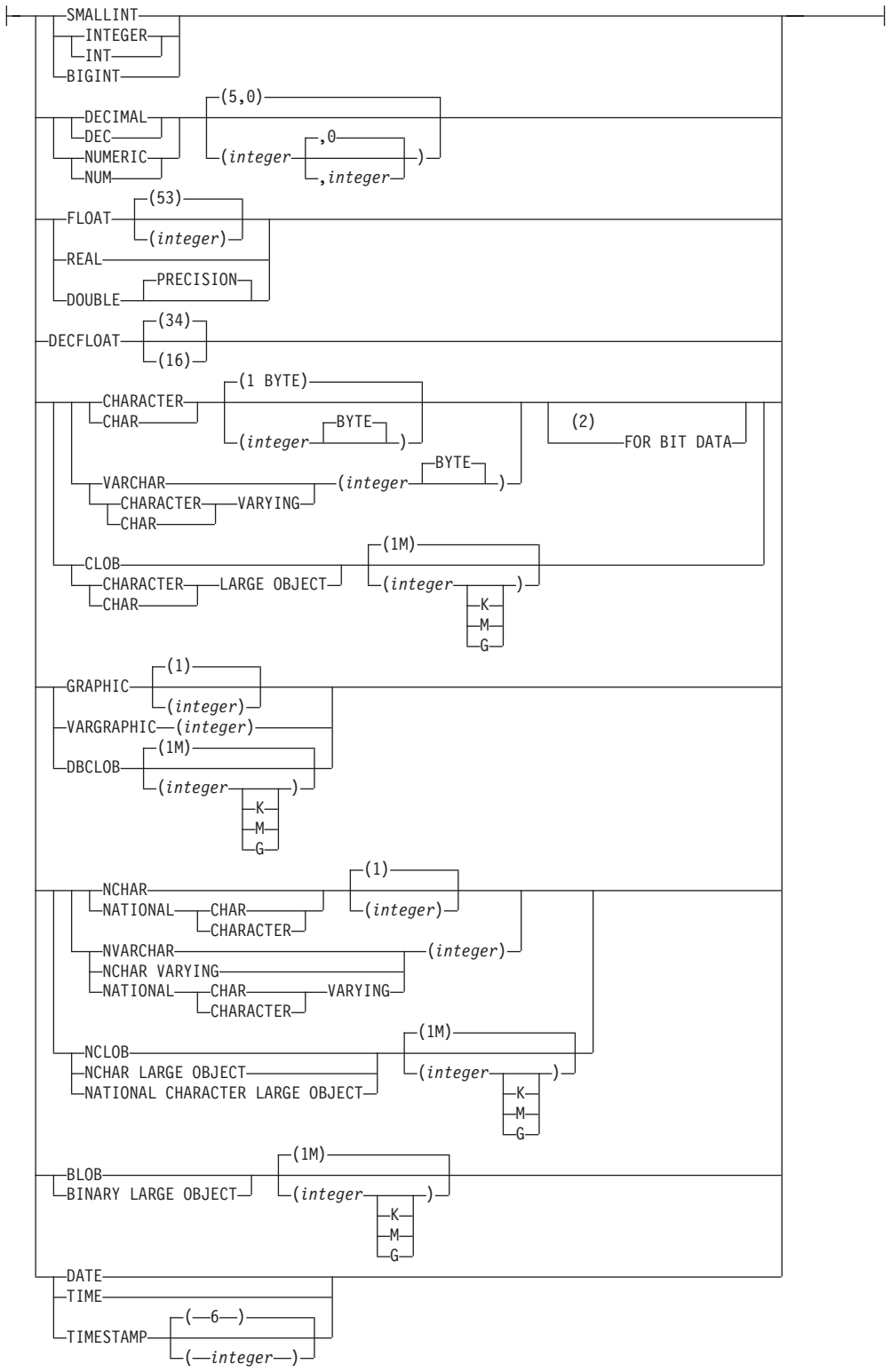


column-definition:

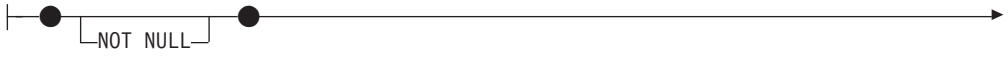


data-type:

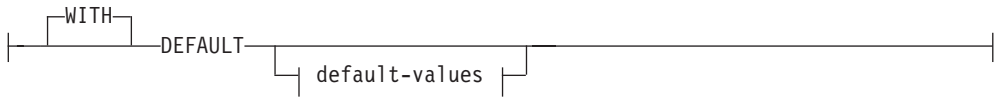
built-in-type:



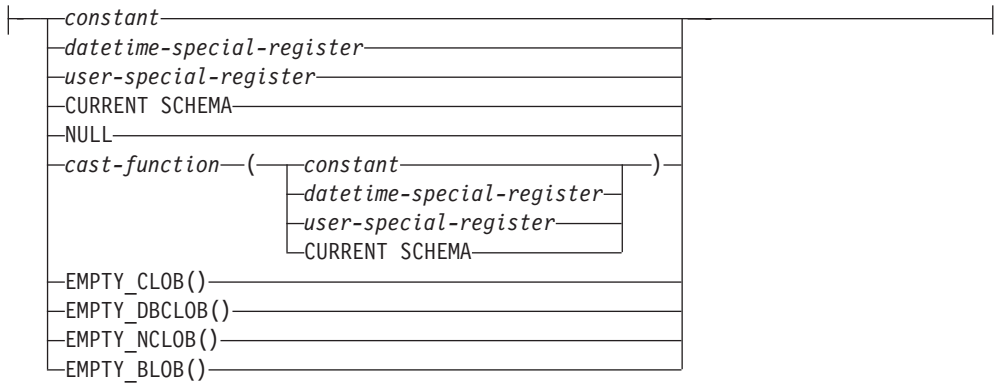
column-options:



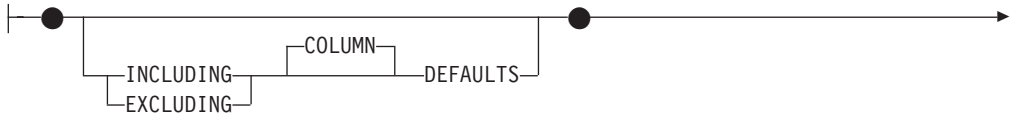
default-clause:

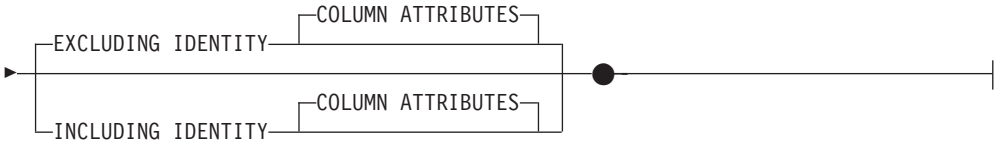


default-values:

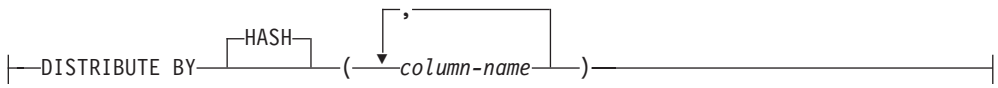


copy-options:





distribution-clause:



Notes:

1 The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type.

2 The FOR BIT DATA clause can be specified in any order with the other column constraints that follow.

Description

table-name
Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname, or alias described in the catalog. If a two-part name is specified, the schema name cannot begin with 'SYS' (SQLSTATE 42939).

column-definition
Defines the attributes of a column of the temporary table.

column-name
Names a column of the table. The name cannot be qualified, and the same name cannot be used for more than one column of the table (SQLSTATE 42711).

A table can have the following:

- A 4K page size with a maximum of 500 columns, where the byte counts of the columns must not be greater than 4 005.
- An 8K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 8 101.
- A 16K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 16 293.
- A 32K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 32 677.

A created temporary table cannot have a row-begin column, row-end column, or a transaction-start-ID column.

For more details, see “Row Size” in “CREATE TABLE”.

data-type
Specifies the data type of the column

built-in-type
Specifies a built-in data type. See “CREATE TABLE” for a description of *built-in-type*.

An XML and SYSPROC.DB2SECURITYLABEL data type cannot be specified for a created temporary table.

distinct-type-name
For a user-defined type that is a distinct type. If a distinct type name is specified without a schema name, the distinct type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a distinct type, then the data type of the column is the distinct type. The length and the scale of the column are respectively the length and the scale of the source type of the distinct type. The distinct type for a column cannot have any data type constraints and the source type cannot be an anchored data type (SQLSTATE 428H2).

column-options
Defines additional options related to the columns of the table.

NOT NULL
Prevents the column from containing null values. For specification of null values, see NOT NULL in “CREATE TABLE”.

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in "ALTER TABLE".

If the column is based on a column of a typed table, a specific default value must be specified when defining a default. A default value cannot be specified for the object identifier column of a typed table (SQLSTATE 42997).

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

If a column is defined using a structured type, the *default-clause* cannot be specified (SQLSTATE 42842).

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

default-values
Specific types of default values that can be specified are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment
- not be a floating-point constant unless the column is defined with a floating-point data type
- be a numeric constant or a decimal floating-point special value if the data type of the column is a decimal floating-point. Floating-point constants are first interpreted as DOUBLE and then converted to decimal floating-point if the target column is DECFLOAT. For DECFLOAT(16) columns, decimal constants having precision greater than 16 digits will be rounded using the rounding modes specified by the CURRENT DECFLOAT ROUNDING MODE special register.
- not have nonzero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time

of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL may be specified within the same column definition but will result in an error on any attempt to set the column to the default value.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

EMPTY_CLOB(), EMPTY_DBCLOB(), or EMPTY_BLOB()
Specifies a zero-length string as the default for the column. The column must have the data type that corresponds to the result data type of the function.

If the value specified is not valid, an error is returned (SQLSTATE 42894).

IDENTITY and *identity-options*
For specification of identity columns, see IDENTITY and *identity-options* in "CREATE TABLE".

LIKE *table-name1* **or** *view-name* **or** *nickname*
Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name1*), view (*view-name*), or nickname (*nickname*). The name specified after LIKE must identify a table, view, or nickname that exists in the catalog, or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC). A protected table cannot be specified (SQLSTATE 42962). A table that has a column defined as IMPLICITLY HIDDEN cannot be specified (SQLSTATE 560AE).

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table (including implicitly hidden columns), view, or nickname. The implicit definition depends on what is identified after LIKE.

- If a table is identified, then the implicit definition includes the column name, data type and nullability characteristic of each of the columns of *table-name1*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*. The data types of the view columns must be data types that are valid for columns of a table.
- If a nickname is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each column of *nickname*.

Column default and identity column attributes may be included or excluded, based on the *copy-attributes* clauses. The implicit definition does not include any other attributes of the identified table, view, or nickname. Thus the new table does not have any unique constraints, foreign key constraints, triggers, indexes, table partitioning keys, or distribution keys. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

When a table is identified in the LIKE clause and that table contains a ROW CHANGE TIMESTAMP column, the corresponding column of the new table inherits only the data type of the ROW CHANGE TIMESTAMP column. The new column is not considered to be a generated column.

If row or column level access control (RCAC) is enforced for *table-name1*, RCAC is not inherited by the new table.

AS (*fullselect***) WITH NO DATA**
Specifies that the columns of the table have the same name and description as the columns that would appear in the derived result table of the fullselect if the fullselect were to be executed. The use of AS (*fullselect*) is an implicit definition of *n* columns for the created temporary table, where *n* is the number of columns that would result from the fullselect.

The implicit definition includes the following attributes of the *n* columns (if applicable to the data type):

- Column name
- Data type, length, precision, and scale
- Nullability

The following attributes are not included (the default value and identity attributes can be included by using the *copy-options*):

- Default value
- Identity attributes
- Hidden attribute
- ROW CHANGE TIMESTAMP

The implicit definition does not include any other optional attributes of the tables or views referenced in the fullselect.

Every select list element must have a unique name (SQLSTATE 42711). The AS clause can be used in the select clause to provide unique names. The fullselect must not refer to host variables or include parameter markers. The data types of the result columns of the fullselect must be data types that are valid for columns of a table.

If row or column level access control (RCAC) is enforced for any table that is specified in *fullselect*, RCAC is not cascaded to the new table.

copy-options
These options specify whether to copy additional attributes of the source result table definition (table, view, or fullselect).

INCLUDING COLUMN DEFAULTS
Column defaults for each updatable column of the source result table definition are copied. Columns that are not updatable will not have a default defined in the corresponding column of the created table.

If LIKE *table-name1* is specified, and *table-name1* identifies a base table, created temporary table, or declared temporary table, then INCLUDING COLUMN DEFAULTS is the default.

EXCLUDING COLUMN DEFAULTS
Column defaults are not copied from the source result table definition.

This clause is the default, except when LIKE *table-name* is specified and *table-name* identifies a base table, created temporary table, or declared temporary table.

INCLUDING IDENTITY COLUMN ATTRIBUTES
If available, identity column attributes (START WITH, INCREMENT BY, and CACHE values) are copied from the source's result table definition. It is possible to copy these attributes if the element of the corresponding column in the table, view, or fullselect is the name of a column of a table, or the name of a column of a view which directly or indirectly maps to the column name of a base table or created temporary table with the identity property. In all other cases, the columns of the new temporary table will not get the identity property. For example:

- The select list of the fullselect includes multiple instances of the name of an identity column (that is, selecting the same column more than once)
- The select list of the fullselect includes multiple identity columns (that is, it involves a join)
- The identity column is included in an expression in the select list
- The fullselect includes a set operation (union, except, or intersect).

EXCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are not copied from the source result table definition.

ON COMMIT
Specifies the action taken on the created temporary table when a COMMIT operation is performed. The default is DELETE ROWS.

DELETE ROWS
All rows of the table will be deleted if no WITH HOLD cursor is open on the table.

PRESERVE ROWS
Rows of the table will be preserved.

LOGGED or NOT LOGGED
Specifies whether operations for the table are logged. The default is NOT LOGGED ON ROLLBACK DELETE ROWS.

NOT LOGGED
Specifies that insert, update, or delete operations against the table are not to be logged, but that the creation or dropping of the table is to be logged. During a ROLLBACK (or ROLLBACK TO SAVEPOINT) operation:

- If the table had been created within a unit of work (or savepoint), the table is dropped
- If the table had been dropped within a unit of work (or savepoint), the table is recreated, but without any data

ON ROLLBACK
Specifies the action that is to be taken on the not logged created temporary table when a ROLLBACK (or ROLLBACK TO SAVEPOINT) operation is performed. The default is DELETE ROWS.

DELETE ROWS
If the table data has been changed, all the rows will be deleted.

PRESERVE ROWS
Rows of the table will be preserved.

LOGGED
Specifies that insert, update, or delete operations against the table as well as the creation or dropping of the table are to be logged.

IN *tablespace-name*
Identifies the table space in which the created temporary table will be instantiated. The table space must exist and be a USER TEMPORARY table space (SQLSTATE 42838), over which the authorization ID of the statement has USE privilege (SQLSTATE 42501). If this clause is not specified, a table space for the table is determined by choosing the USER TEMPORARY table space with the smallest sufficient page size over which the authorization ID of the statement has USE privilege. When more than one table space qualifies, preference is given according to who was granted the USE privilege:

1. The authorization ID
2. A group to which the authorization ID belongs
3. PUBLIC

If more than one table space still qualifies, the final choice is made by the database manager. When no USER TEMPORARY table space qualifies, an error is raised (SQLSTATE 42727).

Determination of the table space can change when:

- Table spaces are dropped or created
- USE privileges are granted or revoked

The sufficient page size of a table is determined by either the byte count of the row or the number of columns. For more details, see "Row Size" in "CREATE TABLE".

distribution-clause
Specifies the database partitioning or the way the data is distributed across multiple database partitions.

DISTRIBUTE BY HASH (*column-name***, ...)**
Specifies the use of the default hashing function on the specified columns, called a *distribution key*, as the distribution method across database partitions. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column whose data type is BLOB, CLOB, DBCLOB, XML, distinct type based on any of these types, or structured type can be used as part of a distribution key (SQLSTATE 42962).

If this clause is not specified, and the table resides in a multiple partition database partition group with multiple database partitions, the distribution key is defined as the first column whose data type is valid for a distribution key.

If none of the columns satisfies the requirements for a default distribution key, the table is created without one. Such tables are allowed only in table spaces that are defined on single-partition database partition groups.

For tables in table spaces that are defined on single-partition database partition groups, any collection of columns with data types that are valid for a distribution key can be used to define the distribution key. If this clause is not specified, no distribution key is created.

Notes

- A user temporary table space must exist before a created temporary table can be created (SQLSTATE 42727).
- ***Instantiation and termination***: For the explanations that follow, P denotes a session and T is a created temporary table in the session P:

  - An empty instance of T is created as a result of the first reference to T that is executed in P.
  - Any SQL statement in P can make reference to T and any reference to T in P is a reference to that same instance of T.
  - Assuming that the ON COMMIT DELETE ROWS clause was specified implicitly or explicitly, then when a commit operation terminates a unit of work in P, and there is no open WITH HOLD cursor in P that is dependent on T, the commit includes the operation DELETE FROM T.
  - When a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes a modification to T:
    - If NOT LOGGED was specified, all rows from T are deleted unless ON ROLLBACK PRESERVE ROWS was also specified
    - If NOT LOGGED was not specified, the changes to T are undone
  - If NOT LOGGED was specified and an INSERT, UPDATE or DELETE statement fails during execution (as opposed to a compilation error), all rows from T are deleted.
  - When a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes the creation of T, then the rollback includes the operation DROP TABLE T.
  - If a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes the drop of a created temporary table T, then the rollback will undo the drop of the table. If NOT LOGGED was specified, then the table will also have been emptied.
  - When the application process that referenced T terminates or disconnects from the database, the private instance of T is dropped and its instantiated rows are destroyed.
  - When the connection to the server at which T was referenced terminates, the private instance of T is dropped and its instantiated rows are destroyed.
- ***Restrictions on the use of created temporary tables***: Created temporary tables cannot:
  - Be specified in an ALTER, LOCK, or RENAME statement (SQLSTATE 42995).
  - Be specified in referential constraints (SQLSTATE 42995).
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DEFINITION ONLY can be specified in place of WITH NO DATA
  - The PARTITIONING KEY clause can be specified in place of the DISTRIBUTE BY clause

  The following syntax is accepted as the default behavior:
  - CCSID ASCII
  - CCSID UNICODE

Examples

- *Example 1*: Create a temporary table, CURRENTMAP. Name two columns, CODE and MEANING, both of which cannot contain nulls. CODE contains numeric data and MEANING has character data.

```
CREATE GLOBAL TEMPORARY TABLE CURRENTMAP
   (CODE        INTEGER      NOT NULL,
    MEANING     VARCHAR(254) NOT NULL)
```

- *Example 2*: Create a temporary table, TMPDEPT.

```
CREATE GLOBAL TEMPORARY TABLE TMPDEPT
   (TMPDEPTNO   CHAR(3)      NOT NULL,
    TMPDEPTNAME VARCHAR(36)  NOT NULL,
    TMPMGRNO    CHAR(6),
    TMPLOCATION CHAR(16) )
```

CREATE HISTOGRAM TEMPLATE

The CREATE HISTOGRAM TEMPLATE statement defines a template describing the type of histogram that can be used to override one or more of the default histograms of a service class or a work class.

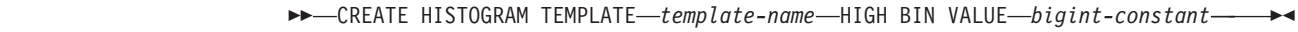
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

Syntax



Description

template-name
Names the histogram template. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must not identify an existing histogram template at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

HIGH BIN VALUE *bigint-constant*
Specifies the top value of the second to last bin (the last bin has an unbounded top value). The units depend on how the histogram is used. The maximum value is 268 435 456.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (WORK ACTION SET)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (WORK CLASS SET)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.

Example

Create a histogram template named LIFETIMETEMP on service class PAYROLL in service superclass ADMIN that will override the default activity lifetime histogram template with a new high bin value of 90 000, which represents 90 000 milliseconds. This will produce a histogram with exponentially increasing bin ranges, ending with a bin whose range is 90 000 to infinity.

```
CREATE HISTOGRAM TEMPLATE LIFETIMETEMP
  HIGH BIN VALUE 90000

CREATE SERVICE CLASS PAYROLL
  UNDER ADMIN ACTIVITY LIFETIME HISTOGRAM TEMPLATE LIFETIMETEMP
```

CREATE INDEX

The CREATE INDEX statement is used to define an index on a DB2 table. An index can be defined on XML data, or on relational data. The CREATE INDEX statement is also used to create an index specification (metadata that indicates to the optimizer that a data source table has an index).

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

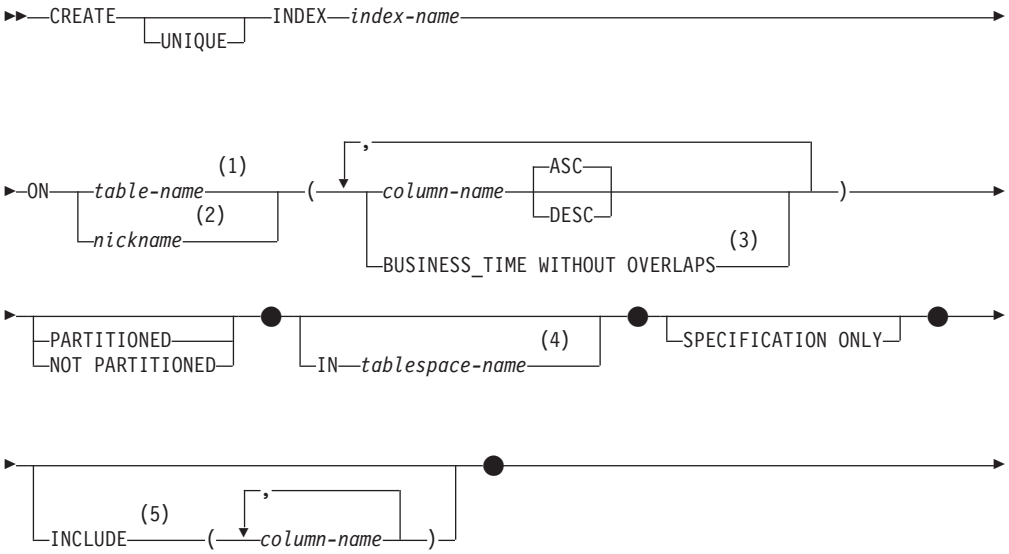
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

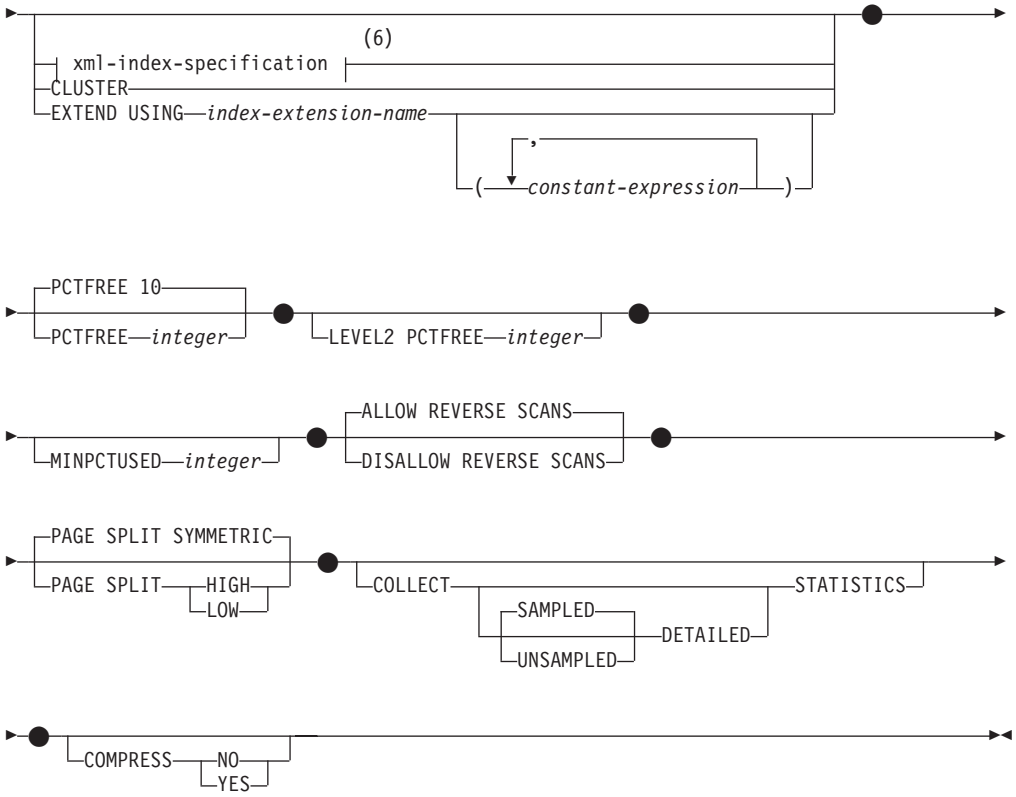
- One of:
  - CONTROL privilege on the table or nickname on which the index is defined
  - INDEX privilege on the table or nickname on which the index is defined

  and one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist
  - CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema
- DBADM authority

No explicit privilege is required to create an index on a declared temporary table.

Syntax





Notes:

1 In a federated system, *table-name* must identify a table in the federated database. It cannot identify a data source table.

2 If *nickname* is specified, the CREATE INDEX statement creates an index specification. In this case, INCLUDE, *xml-index-specification*, CLUSTER, EXTEND USING, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, ALLOW REVERSE SCANS, PAGE SPLIT, or COLLECT STATISTICS cannot be specified.

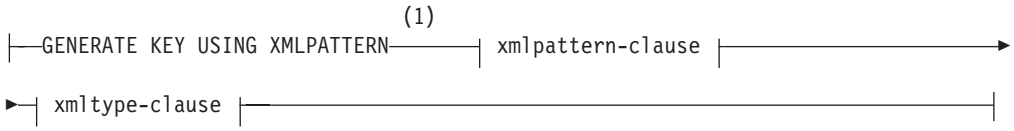
3 The BUSINESS_TIME WITHOUT OVERLAPS clause can be specified only if UNIQUE is specified.

4 The IN *tablespace-name* clause can be specified only for a nonpartitioned index on a partitioned table.

5 The INCLUDE clause can be specified only if UNIQUE is specified.

6 If *xml-index-specification* is specified, *column-name* DESC, INCLUDE, or CLUSTER cannot be specified.

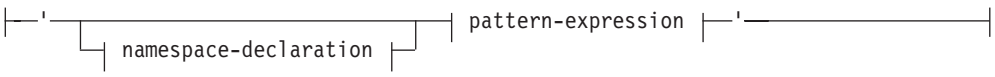
xml-index-specification:



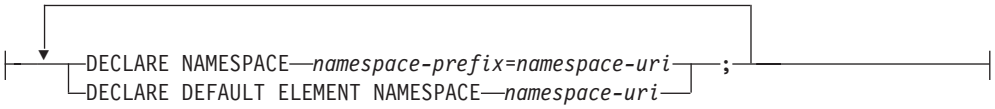
Notes:

1 The alternative syntax GENERATE KEYS USING XMLPATTERN can be used.

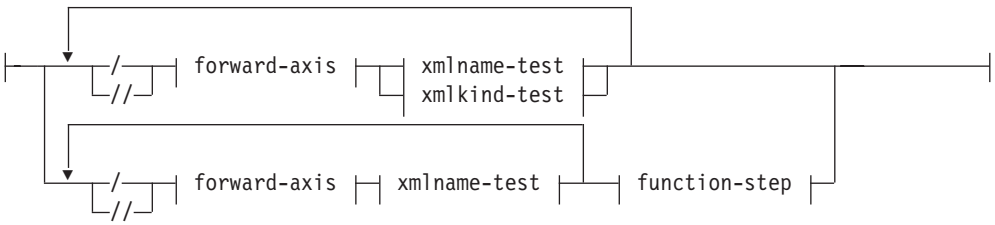
xmlpattern-clause:



namespace-declaration:



pattern-expression:



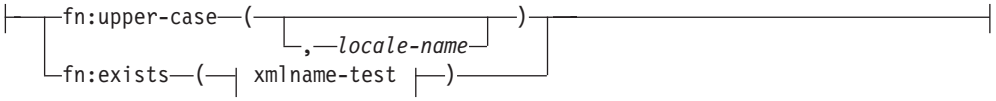
forward-axis:

xmlname-test:

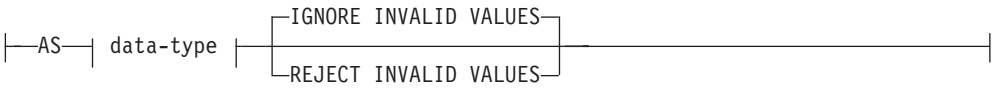
xml-wildcard:

xmlkind-test:

function-step:

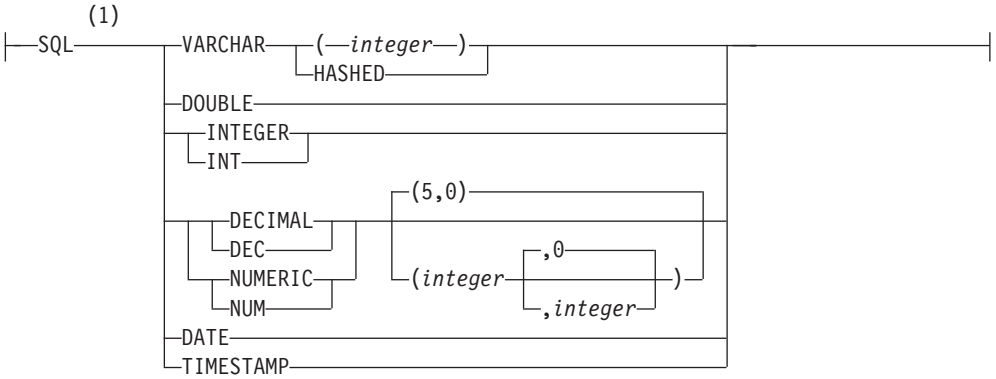


xmltype-clause:



data-type:

sql-data-type:



Notes:

1 If you specify a function name, such as fn:upper-case, at the end of the XML pattern, the supported index data types might be a subset of the index data types shown here. You can check for valid index data types in the description for xmlpattern-clause.

Description

UNIQUE

If ON *table-name* is specified, UNIQUE prevents the table from containing two or more rows with the same value of the index key. The uniqueness is enforced at the end of the SQL statement that updates rows or inserts new rows.

The uniqueness is also checked during the execution of the CREATE INDEX statement. If the table already contains rows with duplicate key values, the index is not created.

If the index is on an XML column (the index is an index over XML data), the uniqueness applies to values with the specified *pattern-expression* for all rows of the table. Uniqueness is enforced on each value after the value has been converted to the specified *sql-data-type*. Because converting to the specified *sql-data-type* might result in a loss of precision or range, or different values might be hashed to the same key value, multiple values that appear to be unique in the XML document might result in duplicate key errors. The uniqueness of character strings depends on XQuery semantics where trailing blanks are significant. Therefore, values that would be duplicates in SQL but differ in trailing blanks are considered unique values in an index over XML data.

When UNIQUE is used, null values are treated as any other values. For example, if the key is a single column that may contain null values, that column may contain no more than one null value.

If the UNIQUE option is specified, and the table has a distribution key, the columns in the index key must be a superset of the distribution key. That is, the columns specified for a unique index key must include all the columns of the distribution key (SQLSTATE 42997).

Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).

If ON *nickname* is specified, UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.

For an index over XML data, UNIQUE can be included only if the context step of the *pattern-expression* specifies a single complete path and does not contain a descendant or descendant-or-self axis, "//", an *xml-wildcard*, *node()*, or *processing-instruction()* (SQLSTATE 429BS).

In a partitioned database environment, the following rules apply to a table with one or more XML columns:

- A distributed table cannot have a unique index over XML data.
- A unique index over XML data is supported only on a table that does not have a distribution key and that is on a single node multi-partition database.
- If a unique index over XML data exists on a table, the table cannot be altered to add a distribution key.

INDEX *index-name*

Names the index or index specification. The name, including the implicit or explicit qualifier, must not identify an index or index specification that is described in the catalog, or an existing index on a declared temporary table (SQLSTATE 42704). The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

The implicit or explicit qualifier for indexes on declared global temporary tables must be SESSION (SQLSTATE 428EK).

ON *table-name* **or** *nickname*

The *table-name* identifies a table on which an index is to be created. The table must be a base table (not a view), a created temporary table, a declared temporary table, a materialized query table that exists at the current server, or a declared temporary table. The name of a declared temporary table must be qualified with SESSION. The *table-name* must not identify a catalog table

(SQLSTATE 42832). If UNIQUE is specified and *table-name* is a typed table, it must not be a subtable (SQLSTATE 429B3).

nickname is the nickname on which an index specification is to be created. The *nickname* references either a data source table whose index is described by the index specification, or a data source view that is based on such a table. The *nickname* must be listed in the catalog.

column-name
For an index, *column-name* identifies a column that is to be part of the index key. For an index specification, *column-name* is the name by which the federated server references a column of a data source table.

Each *column-name* must be an unqualified name that identifies a column of the table. The number of columns plus twice the number of identified periods cannot exceed 64 (SQLSTATE 54008). If *table-name* is a typed table, the number of columns cannot exceed 63 (SQLSTATE 54008). If *table-name* is a subtable, at least one *column-name* must be introduced in the subtable; that is, not inherited from a supertable (SQLSTATE 428DS). No *column-name* can be repeated (SQLSTATE 42711).

The sum of the stored lengths of the specified columns must not be greater than the index key length limit for the page size. For key length limits, see "SQL limits". If *table-name* is a typed table, the index key length limit is further reduced by 4 bytes. Note that this length limit can be reduced even more by system overhead, which varies according to the data type of the column and whether or not the column is nullable. For more information about overhead affecting this limit, see "Byte Counts" in "CREATE TABLE".

Note that this length can be reduced by system overhead, which varies according to the data type of the column and whether it is nullable. For more information on overhead affecting this limit, see "Byte Counts" in "CREATE TABLE".

No LOB column or distinct type column based on a LOB can be used as part of an index, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). A structured type column can only be specified if the EXTEND USING clause is also specified (SQLSTATE 42962). If the EXTEND USING clause is specified, only one column can be specified, and the type of the column must be a structured type or a distinct type that is not based on a LOB (SQLSTATE 42997).

If an index has only one column, and that column has the XML data type, and the GENERATE KEY USING XMLPATTERN clause is also specified, the index is an index over XML data. A column with the XML data type can be specified only if the GENERATE KEY USING XMLPATTERN clause is also specified (SQLSTATE 42962). If the GENERATE KEY USING XMLPATTERN clause is specified, only one column can be specified, and the type of the column must be XML.

ASC
Specifies that index entries are to be kept in ascending order of the column values; this is the default setting. ASC cannot be specified for indexes that are defined with EXTEND USING (SQLSTATE 42601).

DESC
Specifies that index entries are to be kept in descending order of the

column values. DESC cannot be specified for indexes that are defined with EXTEND USING, or if the index is an index over XML data (SQLSTATE 42601).

BUSINESS_TIME WITHOUT OVERLAPS
BUSINESS_TIME WITHOUT OVERLAPS can only be specified for an index defined as UNIQUE (SQLSTATE 428HW) to indicate that for the rest of the specified keys, the values are unique with respect to any period of time. BUSINESS_TIME WITHOUT OVERLAPS can only be specified as the last item in the list. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the end column and begin column of the period BUSINESS_TIME are automatically added to the index key in ascending order and enforce that there are no overlaps in time. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the columns of the BUSINESS_TIME period must not be specified as key columns, as columns in the partitioning key, or as columns in the distribution key (SQLSTATE 428HW).

PARTITIONED
Indicates that a partitioned index should be created. The *table-name* must identify a table defined with data partitions (SQLSTATE 42601).

If the table is partitioned and neither PARTITIONED nor NOT PARTITIONED is specified, the index is created as partitioned (with a few exceptions). A nonpartitioned index is created instead of partitioned index if any of the following situations apply:

- UNIQUE is specified and the index key does not include all the table partitioning key columns.
- A spatial index is created.
- The index is defined over XML data.

A partitioned index with a definition that duplicates the definition of a nonpartitioned index is not considered to be a duplicate index. For more details, see the "Rules" on page 561 section in this topic.

The PARTITIONED keyword cannot be specified for the following indexes:

- An index on a nonpartitioned table (SQLSTATE 42601)
- An index defined over XML data (SQLSTATE 42613)
- A unique index where the index key does not include all the table partitioning key columns (SQLSTATE 42990)
- A spatial index (SQLSTATE 42997)

A partitioned index cannot be created on a partitioned table that has detached dependent tables, for example, MQTs (SQLSTATE 55019).

The table space placement for an index partition of the partitioned index is determined by the following rules:

- If the table being indexed was created using the *partition-tablespace-options* INDEX IN clause of the CREATE TABLE statement, the index partition is created in the table space specified in that INDEX IN clause.
- If the CREATE TABLE statement for the table being indexed did not specify the *partition-tablespace-options* INDEX IN clause, the index partition partitioned index is created in the same table space as the corresponding data partition that it indexes.

The IN clause of the CREATE INDEX statement is not supported for partitioned indexes (SQLSTATE 42601). The *tablespace-clauses* INDEX IN clause of the CREATE TABLE statement is ignored for partitioned indexes. If BUSINESS_TIME WITHOUT OVERLAPS is specified for the index key, the

partitioning key columns must not include the begin or end column of the BUSINESS_TIME period (SQLSTATE 428HW).

NOT PARTITIONED
Indicates that a nonpartitioned index should be created that spans all of the data partitions defined for the table. The *table-name* must identify a table defined with data partitions (SQLSTATE 42601).

A nonpartitioned index with a definition that duplicates the definition of a partitioned index is not considered to be a duplicate index. For more details, see the "Rules" on page 561 section in this topic.

The table space placement for a the nonpartitioned index is determined by the following rules:

- If you specify the IN clause of the CREATE INDEX statement, the nonpartitioned index is placed in the table space specified in that IN clause.
- If you do not specify the IN clause of the CREATE INDEX statement, the following rules determine the table space placement of the nonpartitioned index:
  - If the table being indexed was created using the *tablespace-clauses* INDEX IN clause of the CREATE TABLE statement, the nonpartitioned index is placed in the table space specified in that INDEX IN clause.
  - If the table being indexed was created without using the *tablespace-clauses* INDEX IN clause of the CREATE TABLE statement, the nonpartitioned index is created in the table space of the first visible or attached data partition of the table. The first visible or attached data partition of the table is the first partition in the list of data partitions that are sorted on the basis of range specifications. Also, the authorization ID of the statement is not required to have the USE privilege on the default table space.

IN *tablespace-name*
Specifies the table space in which the nonpartitioned index on a partitioned table is created. This clause cannot be specified for a partitioned index or an index on a nonpartitioned table (SQLSTATE 42601). The specification of a table space specifically for the index overrides a specification made using the INDEX IN clause when the table was created.

The table space specified by *tablespace-name* must be in the same database partition group as the data table spaces for the table and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838); it must be a table space on which the authorization ID of the statement holds the USE privilege.

If the IN clause is not specified, the index is created in the table space that was specified by the INDEX IN clause on the CREATE TABLE statement. If no INDEX IN clause was specified, the table space of the first visible or attached data partition of the table is used. This is the first partition in the list of data partitions that are sorted on the basis of range specifications. If the IN clause is not specified, the authorization ID of the statement is not required to have the USE privilege on the default table space.

SPECIFICATION ONLY
Indicates that this statement will be used to create an index specification that applies to the data source table referenced by *nickname*. SPECIFICATION ONLY must be specified if *nickname* is specified (SQLSTATE 42601). It cannot be specified if *table-name* is specified (SQLSTATE 42601).

If the index specification applies to an index that is unique, DB2 does not verify that the column values in the remote table are unique. If the remote column values are not unique, queries against the nickname that include the index column might return incorrect data or errors.

This clause cannot be used when creating an index on a created temporary table or declared temporary table (SQLSTATE 42995).

INCLUDE
This keyword introduces a clause that specifies additional columns to be appended to the set of index key columns. Any columns included with this clause are not used to enforce uniqueness. These included columns might improve the performance of some queries through index only access. The columns must be distinct from the columns used to enforce uniqueness (SQLSTATE 42711). UNIQUE must be specified when INCLUDE is specified (SQLSTATE 42613). The limits for the number of columns and sum of the length attributes apply to all of the columns in the unique key and in the index.

This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

column-name
Identifies a column that is included in the index but not part of the unique index key. The same rules apply as defined for columns of the unique index key. The keywords ASC or DESC may be specified after *column-name* but have no effect on the order.

INCLUDE cannot be specified for indexes that are defined with EXTEND USING, if *nickname* is specified, or if the index is defined on an XML column (SQLSTATE 42601).

xml-index-specification
Specifies how index keys are generated from XML documents that are stored in an XML column. *xml-index-specification* cannot be specified if there is more than one index column, or if the column does not have the XML data type.

This clause only applies to XML columns (SQLSTATE 429BS).

GENERATE KEY USING XMLPATTERN *xmlpattern-clause*
Specifies the parts of an XML document that are to be indexed. XML pattern values are the indexed values generated by the *xmlpattern-clause*. List data type nodes are not supported in the index. If a node is qualified by the *xmlpattern-clause* and an XML schema exists that specifies that the node is a list data type, then the list data type node cannot be indexed (SQLSTATE 23526 for CREATE INDEX statements, or SQLSTATE 23525 for INSERT and UPDATE statements).

xmlpattern-clause
Contains a pattern expression that identifies the nodes that are to be indexed. It consists of an optional *namespace-declaration* and a required *pattern-expression*.

namespace-declaration
If the pattern expression contains qualified names, a *namespace-declaration* must be specified to define namespace prefixes. A default namespace can be defined for unqualified names.

DECLARE NAMESPACE *namespace-prefix=namespace-uri*
Maps *namespace-prefix*, which is an NCName, to *namespace-uri*,

which is a string literal. The *namespace-declaration* can contain multiple *namespace-prefix*-to-*namespace-uri* mappings. The *namespace-prefix* must be unique within the list of *namespace-declaration* (SQLSTATE 10503).

DECLARE DEFAULT ELEMENT NAMESPACE *namespace-uri*
Declares the default namespace URI for unqualified element names or types. If no default namespace is declared, unqualified names of elements and types are in no namespace. Only one default namespace can be declared (SQLSTATE 10502).

pattern-expression
Specifies the nodes in an XML document that are indexed. The *pattern-expression* can contain pattern-matching characters (*). It is similar to a path expression in XQuery, but supports a subset of the XQuery language that is supported by DB2.

/ (forward slash)
Separates path expression steps.

// (double forward slash)
This is the abbreviated syntax for /descendant-or-self::node()/. You cannot use *// (double forward slash)* if you also specify UNIQUE.

forward-axis

child::
Specifies children of the context node. This is the default, if no other forward axis is specified.

@ Specifies attributes of the context node. This is the abbreviated syntax for attribute::.

attribute::
Specifies attributes of the context node.

descendant::
Specifies the descendants of the context node. You cannot use *descendant::* if you also specify UNIQUE.

self::
Specifies just the context node itself.

descendant-or-self::
Specifies the context node and the descendants of the context node. You cannot use descendant-or-self:: if you also specify UNIQUE.

xmlname-test
Specifies the node name for the step in the path using a qualified XML name (xml-qname) or a wildcard (xml-wildcard).

xml-ncname
An XML name as defined by XML 1.0. It cannot include a colon character.

xml-qname
Specifies a qualified XML name (also known as a QName) that can have two possible forms:

- xml-nsprefix:xml-ncname, where the xml-nsprefix is an xml-ncname that identifies an in-scope namespace
- xml-ncname, which indicates that the default namespace should be applied as the implicit xml-nsprefix

xml-wildcard
Specifies an xml-qname as a wildcard that can have three possible forms:

- * (a single asterisk character) indicates any xml-qname
- *xml-nsprefix*:* indicates any xml-ncname within the specified namespace
- *:*xml-ncname* indicates a specific XML name in any in-scope namespace

You cannot use *xml-wildcard* in the context step of a pattern expression if you also specify UNIQUE.

xmlkind-test
Use these options to specify what types of nodes you pattern match. The following options are available to you:

node()
Matches any node. You cannot use *node()* if you also specify UNIQUE.

text()
Matches any text node.

comment()
Matches any comment node.

processing-instruction()
Matches any processing instruction node. You cannot use processing-instruction() if you also specify UNIQUE.

function-step
Use these function calls to specify indexes with special properties, such as case insensitivity. Only one function step is allowed per XMLPATTERN clause. Function steps can be applied only on elements or attributes. No *xmlkind-test* option can be placed immediately before the function step. The function cannot be used in the middle of the XMLPATTERN, and must appear only in the final step. Currently, only the fn:upper-case and fn:exists functions are supported.

Note that instead of specifying the prefix fn: for the function name, you can specify another valid namespace, or you can omit fn: entirely.

fn:upper-case
Force the index values to be stored in the uppercase form. The first parameter of fn:upper-case is mandatory, and must be a context item expression (' . '); the second parameter is optional, and is the locale. If fn:upper-case appears in the pattern, VARCHAR and VARCHAR HASHED are the only index types supported.

fn:exists
Check for the existence of an element or attribute item in the XML document. If the item exists, this predicate returns

true. The parameter of fn:exists is mandatory, and must be an element or attribute. If this function is used in the index path, the index type must be defined as VARCHAR(1).

xmltype-clause

AS *data-type*
Specifies the data type to which indexed values are converted before they are stored. Values are converted to the index XML data type that corresponds to the specified index SQL data type.

Table 17. Corresponding index data types

Index XML data type	Index SQL data type
xs:string	VARCHAR(*integer*), VARCHAR HASHED
xs:double	DOUBLE
xs:int	INTEGER
xs:decimal	DECIMAL
xs:date	DATE
xs:dateTime	TIMESTAMP

For VARCHAR(*integer*) and VARCHAR HASHED, the value is converted to an xs:string value using the XQuery function fn:string. The length attribute of VARCHAR(*integer*) is applied as a constraint to the resulting xs:string value. An index SQL data type of VARCHAR HASHED applies a hash algorithm to the resulting xs:string value to generate a hash code that is inserted into the index.

For indexes using the data types DOUBLE, DATE, INTEGER, DECIMAL, and TIMESTAMP, the value is converted to the index XML data type using the XQuery cast expression.

If the index is unique, the uniqueness of the value is enforced after the value is converted to the indexed type.

data-type
The following data type is supported:

sql-data-type
Supported SQL data types are:

VARCHAR(*integer***)**
If this form of VARCHAR is specified, DB2 uses *integer* as a constraint. If document nodes that are to be indexed have values that are longer than *integer*, the documents are not inserted into the table if the index already exists. If the index does not exist, the index is not created. *integer* is a value between 1 and a page size-dependent maximum. Table 18 shows the maximum value for each page size.

Table 18. Maximum length of document nodes by page size

Page size	Maximum length of document node (bytes)
4KB	817
8KB	1841
16KB	3889

Table 18. Maximum length of document nodes by page size (continued)

Page size	Maximum length of document node (bytes)
32KB	7985

XQuery semantics are used for string comparisons, where trailing blanks are significant. This differs from SQL semantics, where trailing blanks are insignificant during comparisons.

VARCHAR HASHED
Specify VARCHAR HASHED to handle indexing of arbitrary length character strings. The length of an indexed string has no limit. DB2 generates an eight-byte hash code over the entire string. Indexes that use these hashed character strings can be used only for equality lookups. XQuery semantics are used for string equality comparisons, where trailing blanks are significant. This differs from SQL semantics, where trailing blanks are insignificant during comparisons. The hash on the string preserves XQuery semantics for equality and not SQL semantics.

DOUBLE
Specifies that the data type DOUBLE is used for indexing numeric values. Unbounded decimal types and 64 bit integers may lose precision when they are stored as a DOUBLE value. The values for DOUBLE may include the special numeric values *NaN*, *INF*, *-INF*, *+0*, and *-0*, even though the SQL data type DOUBLE itself does not support these values.

INTEGER
Specifies that the data type INTEGER is used for indexing XML values. Note that the XML schema data type xs:integer allows a greater range of values than does the integer SQL data type. If an out-of-range value is encountered, an error is returned. If a value conforms to the lexical format of xs:double but does not conform to the lexical format of xs:int, such as 3.5, 3.0, or 3E1, an error is also returned.

DECIMAL(*integer*, *integer*)
Specifies that the data type DECIMAL is used for indexing XML values. The DECIMAL type takes two parameters, *precision* and *scale*. The first parameter, *precision*, is an integer constant with a value in the range of 1 to 31 that specifies the total number of digits. The second parameter, *scale*, is an integer constant that is greater than or equal to zero, and less than or equal to precision. The *scale* specifies the number of digits to the right of the decimal point.

Digits are not truncated from the end of a decimal number. An error is returned if the number of digits to the right of the decimal separator character is greater than the scale. Also, an error is returned if the number

of significant digits to the left of the decimal character (the whole part of the number) is greater than precision.

DATE
Specifies that the data type DATE is used for indexing XML values. Note that the XML schema data type for xs:date allows greater range of values than the DB2 pureXML® xs:date data type that corresponds to the SQL data type. If an out-of-range value is encountered, an error is returned.

TIMESTAMP
Specifies that the data type TIMESTAMP is used for indexing XML values. Note that the XML schema data type for xs:dateTime allows greater range of values and fractional seconds precision than the DB2 pureXML xs:dateTime data type that corresponds to the SQL data type. If an out-of range value is encountered, an error is returned.

IGNORE INVALID VALUES
Specifies that XML pattern values that are invalid lexical forms for the target index XML data type are ignored and that the corresponding values in the stored XML documents are not indexed by the CREATE INDEX statement. By default, invalid values are ignored. During insert and update operations, the invalid XML pattern values are not indexed, but XML documents are still inserted into the table. No error or warning is raised, because specifying these data types is not a constraint on the XML pattern values (XQuery expressions that search for the specific XML index data type will not consider these values).

The rules for what XML pattern values can be ignored are determined by the specified SQL data type.

- If the SQL data type is VARCHAR(*integer*) or VARCHAR HASHED, XML pattern values are never ignored since any sequence of characters is valid.
- If the SQL data type is DOUBLE, DECIMAL, or INTEGER, any XML pattern value that does not conform to the lexical format of the XML data type xs:double is ignored. If the SQL data type is DECIMAL or INTEGER and the XML pattern value conforms to the lexical format of the XML data type xs:double but not to the lexical format of xs:decimal or xs:int, respectively, an error is returned. For example, if the SQL data type is INTEGER, the XML pattern values of 3.5, 3.0, and 3e0 conform to the lexical format of xs:double but return an error (SQLSTATE 23525) because they do not conform to the lexical format of xs:int. XML pattern values such as 'A123' or 'hello' are ignored for the same index.
- If the SQL data type is a datetime data type, any XML pattern value that does not conform to the lexical format of the corresponding XML data type (xs:date or xs:dateTime) is ignored.

If an XML pattern value does conform to the appropriate lexical format, an error is returned if the value is outside the value space for the data type or exceeds the maximum length or precision and

scale of the specified SQL data type. If the index does not exist, the index is not created (SQLSTATE 23526).

REJECT INVALID VALUES
All XML pattern values must be valid in the context of the lexical definition of the index XML data type. In addition the value must be in the range of the value space of the index XML data type. See the Related reference section, later, for links to details on the lexical definition and value space for each data type. For example, when you specify the REJECT INVALID VALUES clause, if you create an index of INTEGER type, XML pattern values such as 3.5, 3.0, 3e0, 'A123' and 'hello' will return an error (SQLSTATE 23525). XML data is not inserted or updated in the table if the index already exists (SQLSTATE 23525). If the index does not exist, the index is not created (SQLSTATE 23526).

CLUSTER
Specifies that the index is the clustering index of the table. The cluster factor of a clustering index is maintained or improved dynamically as data is inserted into the associated table, by attempting to insert new rows physically close to the rows for which the key values of this index are in the same range. Only one clustering index may exist for a table so CLUSTER may not be specified if it was used in the definition of any existing index on the table (SQLSTATE 55012). A clustering index may not be created on a table that is defined to use append mode (SQLSTATE 428D8).

CLUSTER is disallowed if *nickname* is specified, or if the index is an index over XML data (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995) or range-clustered tables (SQLSTATE 429BG).

EXTEND USING *index-extension-name*
Names the *index-extension* used to manage this index. If this clause is specified, then there must be only one *column-name* specified and that column must be a structured type or a distinct type (SQLSTATE 42997). The *index-extension-name* must name an index extension described in the catalog (SQLSTATE 42704). For a distinct type, the column must exactly match the type of the corresponding source key parameter in the index extension. For a structured type column, the type of the corresponding source key parameter must be the same type or a supertype of the column type (SQLSTATE 428E0).

This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

This clause cannot be used in a DB2 pureScale environment (SQLSTATE 56038).

constant-expression
Identifies values for any required arguments for the index extension. Each expression must be a constant value with a data type that exactly matches the defined data type of the corresponding index extension parameters, including length or precision, and scale (SQLSTATE 428E0). This clause must not exceed 32 768 bytes in length in the database code page (SQLSTATE 22001).

PCTFREE *integer*
Specifies what percentage of each index page to leave as free space when building the index. The first entry in a page is added without restriction. When additional entries are placed in an index page at least *integer* percent of free

space is left on each page. The value of *integer* can range from 0 to 99. If a value greater than 10 is specified, only 10 percent free space will be left in non-leaf pages.

If an explicit value for PCTFREE is not provided, and if **DB2_INDEX_PCTFREE_DEFAULT** is not set, then PCTFREE will have a default value of 10.

PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

LEVEL2 PCTFREE *integer*
Specifies what percentage of each index level 2 page to leave as free space when building the index. The value of *integer* can range from 0 to 99. If LEVEL2 PCTFREE is not set, a minimum of 10 or PCTFREE percent of free space is left on all non-leaf pages. If LEVEL2 PCTFREE is set, *integer* percent of free space is left on level 2 intermediate pages, and a minimum of 10 or *integer* percent of free space is left on level 3 and higher intermediate pages.

LEVEL2 PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

MINPCTUSED *integer*
Indicates whether index leaf pages are merged online, and the threshold for the minimum percentage of space used on an index leaf page. If, after a key is removed from an index leaf page, the percentage of space used on the page is at or below *integer* percent, an attempt is made to merge the remaining keys on this page with those of a neighboring page. If there is sufficient space on one of these pages, the merge is performed and one of the pages is deleted. The value of *integer* can be from 0 to 99. A value of 50 or below is recommended for performance reasons. Specifying this option will have an impact on update and delete performance. Merging is only done during update and delete operations when an exclusive table lock is held. If an exclusive table lock does not exist, keys are marked as pseudo deleted during update and delete operations, and no merging is done. Consider using the CLEANUP ONLY ALL option of REORG INDEXES to merge leaf pages instead of using the MINPCTUSED option of CREATE INDEX.

MINPCTUSED is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with created temporary tables or declared temporary tables (SQLSTATE 42995).

DISALLOW REVERSE SCANS
Specifies that an index only supports forward scans or scanning of the index in the order that was defined at index creation time.

DISALLOW REVERSE SCANS cannot be specified together with *nickname* (SQLSTATE 42601).

ALLOW REVERSE SCANS
Specifies that an index can support both forward and reverse scans; that is, scanning of the index in the order that was defined at index creation time, and scanning in the opposite order.

ALLOW REVERSE SCANS cannot be specified together with *nickname* (SQLSTATE 42601).

PAGE SPLIT
Specifies an index split behavior. The default is SYMMETRIC.

SYMMETRIC
Specifies that pages are to be split roughly in the middle.

HIGH
Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a particular pattern. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that increase with each insertion.

LOW
Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a particular pattern. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that decrease with each insertion.

COLLECT STATISTICS
Specifies that basic index statistics are to be collected during index creation.

SAMPLED
Specifies that a sampling technique is to be used when processing index entries to collect extended index statistics. This option is used to balance performance considerations with the need for accuracy of the statistics. This option is the default when DETAILED is specified immediately following the keyword COLLECT.

UNSAMPLED
Specifies that sampling is not to be used when processing index entries to collect extended index statistics. Instead, each index entry is examined individually. This option can significantly increase CPU and memory consumption.

DETAILED
Specifies that extended index statistics (CLUSTERFACTOR and PAGE_FETCH_PAIRS) are also to be collected during index creation.

COMPRESS
Specifies whether index compression is enabled. By default, index compression will be enabled if data row compression is enabled; index compression will be disabled if data row compression is disabled. This option can be used to override the default behavior. COMPRESS is disallowed if *nickname* is specified (SQLSTATE 42601).

YES
Specifies that index compression is enabled. Insert and update operations on the index will be subject to compression.

NO Specifies that index compression is disabled.

Rules

- The CREATE INDEX statement fails (SQLSTATE 01550) when attempting to create an index that matches an existing index.

  A number of factors are used to determine if two indexes match. These factors are combined in various different ways into the rules that determine if two indexes match. The following factors are used to determine if two indexes match:

1. The sets of index columns, including any INCLUDE columns, are the same in both indexes.
2. The ordering of index key columns, including any INCLUDE columns, is the same in both indexes.
3. The key columns of the new index are the same or a superset of the key columns in the existing index.
4. The ordering attributes of the columns are the same in both indexes.
5. The existing index is unique.
6. Both indexes are non-unique.

The following combinations of these factors form the rules that determine when two indexes are considered duplicates:

– 1 + 2 + 4 + 5
– 1 + 2 + 4 + 6
– 1 + 2 + 3 + 5

Exceptions:

– If one of the compared indexes is partitioned and the other of the compared indexes is nonpartitioned, the indexes are not considered duplicates if the indexes have different names, even if other matching index conditions are met.
– For indexes over XML data, the index descriptions are not considered duplicates if the index names are different, even if the indexed XML column, the XML patterns, and the data type, including its options, are identical.

- Unique indexes on system-maintained MQTs are not supported (SQLSTATE 42809).
- The COLLECT STATISTICS options are not supported if a nickname is specified (SQLSTATE 42601).

Notes

- Concurrent read/write access during index creation, and default index creation behavior differs for indexes on nonpartitioned tables, nonpartitioned indexes, partitioned indexes, and indexes in a DB2 pureScale environment:
  – For nonpartitioned indexes, concurrent read/write access to the table is permitted while an index is being created, except when the EXTEND USING clause is specified. Once the index has been built, changes that were made to the table during index creation time are forward-fitted to the new index. Write access to the table is then briefly blocked while index creation completes, after which the new index becomes available.
  – For partitioned indexes, concurrent read/write access to the table is permitted while an index is being created, except when the EXTEND USING clause is specified. Once the index partition has been built, changes that were made to the partition during creation time of that index partition are forward-fitted to the new index partition. Write access to the data partition is then blocked while index creation completes on the remaining data partitions. After the index partition for the last data partition is built and the transaction is committed, all data partitions are available for read and write.
  – In a DB2 pureScale environment, concurrent read access is the default behavior. Concurrent write access is not allowed during index creation.

To circumvent this default behavior, use the LOCK TABLE statement to explicitly lock the table before issuing a CREATE INDEX statement. (The table can be locked in either SHARE or EXCLUSIVE mode, depending on whether read access is to be allowed.)

- If the named table already contains data, CREATE INDEX creates the index entries for it. If the table does not yet contain data, CREATE INDEX creates a description of the index; the index entries are created when data is inserted into the table.
- b
- Once the index is created and data is loaded into the table, it is advisable to issue the RUNSTATS command. The RUNSTATS command updates statistics collected on the database tables, columns, and indexes. These statistics are used to determine the optimal access path to the tables. By issuing the RUNSTATS command, the database manager can determine the characteristics of the new index. If data has been loaded before the CREATE INDEX statement is issued, it is recommended that the COLLECT STATISTICS option on the CREATE INDEX statement be used as an alternative to the RUNSTATS command.
- Creating an index with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- The optimizer can recommend indexes before creating the actual index.
- If an index specification is being defined for a data source table that has an index, the name of the index specification does not have to match the name of the index.
- The optimizer uses index specifications to improve access to the data source tables that the specifications apply to.
- ***Collecting index statistics***: The UNSAMPLED DETAILED option is available to change the way index statistics are collected. However, it should be used only in cases where it is clear that DETAILED does not yield accurate statistics.
- ***Syntax alternatives***: The following syntax is tolerated and ignored:
  - CLOSE
  - DEFINE
  - FREEPAGE
  - GBPCACHE
  - PIECESIZE
  - TYPE 2
  - using-block

  The following syntax is accepted as the default behavior:
  - COPY NO
  - DEFER NO

Examples

- *Example 1:* Create an index named UNIQUE_NAM on the PROJECT table. The purpose of the index is to ensure that there are not two entries in the table with the same value for project name (PROJNAME). The index entries are to be in ascending order.

```
CREATE UNIQUE INDEX UNIQUE_NAM
   ON PROJECT(PROJNAME)
```

- *Example 2:* Create an index named JOB_BY_DPT on the EMPLOYEE table. Arrange the index entries in ascending order by job title (JOB) within each department (WORKDEPT).

```
CREATE INDEX JOB_BY_DPT
   ON EMPLOYEE (WORKDEPT, JOB)
```

- *Example 3:* The nickname EMPLOYEE references a data source table called CURRENT_EMP. After this nickname was created, an index was defined on CURRENT_EMP. The columns chosen for the index key were WORKDEBT and JOB. Create an index specification that describes this index. Through this specification, the optimizer will know that the index exists and what its key is. With this information, the optimizer can improve its strategy to access the table.

```
CREATE UNIQUE INDEX JOB_BY_DEPT
  ON EMPLOYEE (WORKDEPT, JOB)
  SPECIFICATION ONLY
```

- *Example 4:* Create an extended index type named SPATIAL_INDEX on a structured type column location. The description in index extension GRID_EXTENSION is used to maintain SPATIAL_INDEX. The literal is given to GRID_EXTENSION to create the index grid size.

```
CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
  EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
```

- *Example 5:* Create an index named IDX1 on a table named TAB1, and collect basic index statistics on index IDX1.

```
CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
```

- *Example 6:* Create an index named IDX2 on a table named TAB1, and collect detailed index statistics on index IDX2.

```
CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
```

- *Example 7:* Create an index named IDX3 on a table named TAB1, and collect detailed index statistics on index IDX3 using sampling.

```
CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
```

- *Example 8:* Create a unique index named A_IDX on a partitioned table named MYNUMBERDATA in table space IDX_TBSP.

```
CREATE UNIQUE INDEX A_IDX ON MYNUMBERDATA (A) IN IDX_TBSP
```

- *Example 9:* Create a non-unique index named B_IDX on a partitioned table named MYNUMBERDATA in table space IDX_TBSP.

```
CREATE INDEX B_IDX ON MYNUMBERDATA (B)
  NOT PARTITIONED IN IDX_TBSP
```

- *Example 10:* Create an index over XML data on a table named COMPANYINFO, which contains an XML column named COMPANYDOCS. The XML column COMPANYDOCS contains a large number of XML documents similar to the one below:

```
<company name="Company1">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">
      Finance
    </dept>
  </emp>
</company>
```

Users of the COMPANYINFO table often need to retrieve employee information using the employee ID. An index like the following one can make that retrieval more efficient.

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'
    AS SQL DOUBLE
```

- *Example 11:* The following index is logically equivalent to the index created in the previous example, except that it uses unabbreviated syntax.

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
  GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
    AS SQL DOUBLE
```

- *Example 12:* Create an index on a column named DOC, indexing only the book title as a VARCHAR(100). Because the book title should be unique across all books, the index must be unique.

```
CREATE UNIQUE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN '/book/title'
    AS SQL VARCHAR(100)
```

- *Example 13:* Create an index on a column named DOC, indexing the chapter number as a DOUBLE. This example includes namespace declarations.

```
CREATE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN
    'declare namespace b="http://www.example.com/book/";
      declare namespace c="http://acme.org/chapters";
        /b:book/c:chapter/@number'
    AS SQL DOUBLE
```

- *Example 14:* Create a unique index named IDXPROJEST on table PROJECT and include column PRSTAFF to allow index-only access of the estimated mean staffing information.

```
CREATE UNIQUE INDEX IDXPROJEST ON PROJECT (PROJNO) INCLUDE (PRSTAFF)
```

CREATE INDEX EXTENSION

The CREATE INDEX EXTENSION statement defines an extension object for use with indexes on tables that have structured type or distinct type columns.

Invocation

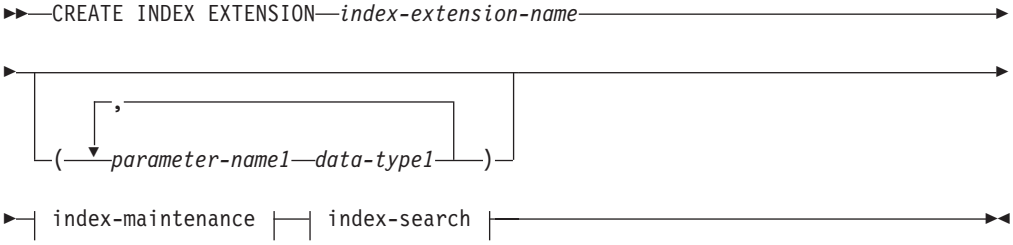
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

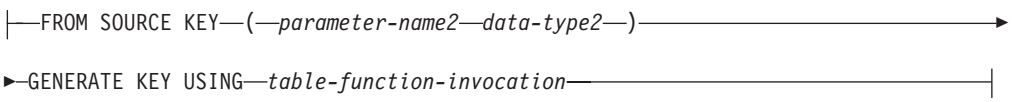
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the schema name of the index extension does not refer to an existing schema
- CREATEIN privilege on the schema, if the schema name of the index extension refers to an existing schema
- DBADM authority

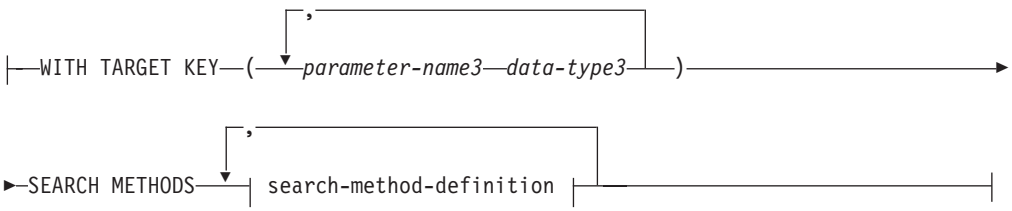
Syntax



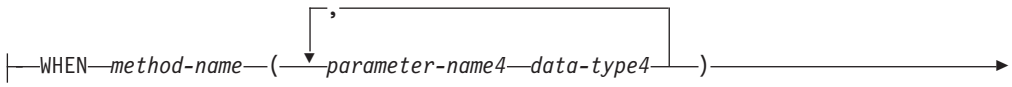
index-maintenance:

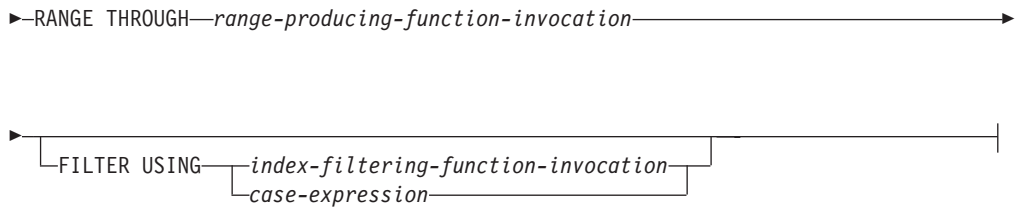


index-search:



search-method-definition:





Description

index-extension-name
Names the index extension. The name, including the implicit or explicit qualifier, must not identify an index extension described in the catalog. If a two-part *index-extension-name* is specified, the schema name cannot begin with 'SYS'; otherwise, an error is returned (SQLSTATE 42939).

parameter-name1
Identifies a parameter that is passed to the index extension at CREATE INDEX time to define the actual behavior of this index extension. The parameter that is passed to the index extension is called an *instance parameter*, because that value defines a new instance of an index extension.

parameter-name1 must be unique within the definition of the index extension. No more than 90 parameters are allowed. If this limit is exceeded, an error (SQLSTATE 54023) is returned.

data-type1
Specifies the data type of each parameter. One entry in the list must be specified for each parameter that the index extension will expect to receive. The only SQL data types that can be specified are those that can be used as constants, such as VARCHAR, INTEGER, DECIMAL, DOUBLE, or VARGRAPHIC (SQLSTATE 429B5). The decimal floating-point data type cannot be specified (SQLSTATE 429B5). The parameter value that is received by the index extension at CREATE INDEX must match *data-type1* exactly, including length, precision, and scale (SQLSTATE 428E0).

index-maintenance
Specifies how the index keys of a structured or distinct type column are maintained. Index maintenance is the process of transforming the source column to a target key. The transformation process is defined using a table function that has previously been defined in the database.

FROM SOURCE KEY (*parameter-name2 data-type2***)**
Specifies a structured data type or distinct type for the source key column that is supported by this index extension.

parameter-name2
Identifies the parameter that is associated with the source key column. A source key column is the index key column (defined in the CREATE INDEX statement) with the same data type as *data-type2*.

data-type2
Specifies the data type for *parameter-name2*; *data-type2* must be a user-defined structured type or a distinct type that is not sourced on LOB, XML, or DECFLOAT (SQLSTATE 42997). When the index extension is associated with the index at CREATE INDEX time, the data type of the index key column must:

- Exactly match *data-type2* if it is a distinct type; or
- Be the same type or a subtype of *data-type2* if it is a structured type

Otherwise, an error is returned (SQLSTATE 428E0).

GENERATE KEY USING *table-function-invocation*
Specifies how the index key is generated using a user-defined table function. Multiple index entries may be generated for a single source key data value. An index entry cannot be duplicated from a single source key data value (SQLSTATE 22526). The function can use *parameter-name1*, *parameter-name2*, or a constant as arguments. If the data type of *parameter-name2* is a structured data type, only the observer methods of that structured type can be used in its arguments (SQLSTATE 428E3). The output of the GENERATE KEY function must be specified in the TARGET KEY specification. The output of the function can also be used as input for the index filtering function specified on the FILTER USING clause.

The function used in *table-function-invocation* must:

- Resolve to a table function (SQLSTATE 428E4)
- Not be defined with PARAMETER CCSID UNICODE if this database is not a Unicode database (SQLSTATE 428E4)
- Not be defined with LANGUAGE SQL (SQLSTATE 428E4)
- Not be defined with NOT DETERMINISTIC (SQLSTATE 428E4) or EXTERNAL ACTION (SQLSTATE 428E4)
- Be defined with NO SQL (SQLSTATE 428E4)
- Not have a structured data type, LOB or XML (SQLSTATE 428E3) in the data type of the parameters, with the exception of system-generated observer methods
- Not include a subquery (SQLSTATE 428E3)
- Not include an XMLQUERY or XMLEXISTS expression (SQLSTATE 428E3)
- Return columns with data types that follow the restrictions for data types of columns of an index defined without the EXTEND USING clause

If an argument invokes another operation or routine, it must be an observer method (SQLSTATE 428E3).

The definer of the index extension must have EXECUTE privilege on this function.

index-search
Specifies how searching is performed by providing a mapping of the search arguments to search ranges.

WITH TARGET KEY
Specifies the target key parameters that are the output of the key generation function specified on the GENERATE KEY USING clause.

parameter-name3
Identifies the parameter associated with a given target key. *parameter-name3* corresponds to the columns of the RETURNS table as specified in the table function of the GENERATE KEY USING clause. The number of parameters specified must match the number of columns returned by that table function (SQLSTATE 428E2).

data-type3
Specifies the data type for each corresponding *parameter-name3*. *data-type3*

must exactly match the data type of each corresponding output column of the RETURNS table, as specified in the table function of the GENERATE KEY USING clause (SQLSTATE 428E2), including the length, precision, and type.

SEARCH METHODS
Introduces the search methods that are defined for the index.

search-method-definition
Specifies the method details of the index search. It consists of a method name, the search arguments, a range producing function, and an optional index filter function.

WHEN *method-name*
The name of a search method. This is an SQL identifier that relates to the method name specified in the index exploitation rule (found in the PREDICATES clause of a user-defined function). A *search-method-name* can be referenced by only one WHEN clause in the search method definition (SQLSTATE 42713).

parameter-name4
Identifies the parameter of a search argument. These names are for use in the RANGE THROUGH and FILTER USING clauses.

data-type4
The data type associated with a search parameter.

RANGE THROUGH *range-producing-function-invocation*
Specifies an external table function that produces search ranges. This function uses *parameter-name1*, *parameter-name4*, or a constant as arguments and returns a set of search ranges.

The table function used in *range-producing-function-invocation* must:

- Resolve to a table function (SQLSTATE 428E4)
- Not include a subquery (SQLSTATE 428E3) or SQL function (SQLSTATE 428E4) in its arguments
- Not include an XMLQUERY or XMLEXISTS expression in its arguments (SQLSTATE 428E3)
- Not be defined with PARAMETER CCSID UNICODE if this database is not a Unicode database (SQLSTATE 428E4)
- Not be defined with LANGUAGE SQL (SQLSTATE 428E4)
- Not be defined with NOT DETERMINISTIC or EXTERNAL ACTION (SQLSTATE 428E4)
- Be defined with NO SQL (SQLSTATE 428E4)

The number and types of this function's results must relate to the results of the table function specified in the GENERATE KEY USING clause (SQLSTATE 428E1) by:

- Returning up to twice as many columns as returned by the key transformation function
- Having an even number of columns, in which the first half of the return columns defines the start of the range (start key values), and the second half of the return columns defines the end of the range (stop key values)
- Having each start key column with the same type as the corresponding stop key column
- Having the type of each start key column be the same as the corresponding key transformation function column

More precisely, let $a_1:t_1$, ..., $a_n:t_n$ be the function result columns and data types of the key transformation function. The function result columns of the *range-producing-function-invocation* must be $b_1:t_1$, ..., $b_m:t_m$, $c_1:t_1$, ..., $c_m:t_m$, where m <= n and the "b" columns are the start key columns and the "c" columns are the stop key columns.

When the *range-producing-function-invocation* returns a null value as the start or stop key value, the semantics are undefined.

The definer of the index extension must have EXECUTE privilege on this function.

FILTER USING

Allows specification of an external function or a case expression to be used for filtering index entries that were returned after applying the range-producing function.

index-filtering-function-invocation

Specifies an external function to be used for filtering index entries. This function uses the *parameter-name1*, *parameter-name3*, *parameter-name4*, or a constant as arguments (SQLSTATE 42703) and returns an integer (SQLSTATE 428E4). If the value returned is 1, the row corresponding to the index entry is retrieved from the table. Otherwise, the index entry is not considered for further processing.

If not specified, index filtering is not performed.

The function used in the *index-filtering-function-invocation* must:

- Not be defined with PARAMETER CCSID UNICODE if this database is not a Unicode database (SQLSTATE 428E4)
- Not be defined with LANGUAGE SQL (SQLSTATE 429B4)
- Not be defined with NOT DETERMINISTIC or EXTERNAL ACTION (SQLSTATE 42845)
- Be defined with NO SQL (SQLSTATE 428E4)
- Not have a structured data type in the data type of any of the parameters (SQLSTATE 428E3)
- Not include a subquery (SQLSTATE 428E3)
- Not include an XMLQUERY or XMLEXISTS expression (SQLSTATE 428E3)

If an argument invokes another function or method, these rules are also enforced for this nested function or method. However, system-generated observer methods are allowed as arguments to the filter function (or any function or method used as an argument), as long as the argument results in a built-in data type.

The definer of the index extension must have EXECUTE privilege on this function.

case-expression

Specifies a case expression for filtering index entries. Either *parameter-name1*, *parameter-name3*, *parameter-name4*, or a constant (SQLSTATE 42703) can be used in the *searched-when-clause* and *simple-when-clause*. An external function with the rules specified in FILTER USING *index-filtering-function-invocation* may be used in *result-expression*. Any function referenced in the *case-expression* must also conform to the rules listed under *index-filtering-function-invocation*. In addition, subqueries and XMLQUERY or XMLEXISTS expressions cannot be used anywhere else in the *case-expression* (SQLSTATE 428E4). The case expression must return

an integer (SQLSTATE 428E4). A return value of 1 in the *result-expression* means that the index entry is kept; otherwise, the index entry is discarded.

Notes

- Creating an index extension with a schema name that does not already exist will result in the implicit creation of that schema, provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

Example

The following example creates an index extension called *grid_extension* that uses a structured type SHAPE column in a table function called *gridEntry* to generate seven index target keys. This index extension also provides two index search methods to produce search ranges when given a search argument.

```
CREATE INDEX EXTENSION GRID_EXTENSION (LEVELS VARCHAR(20) FOR BIT DATA)
  FROM SOURCE KEY (SHAPECOL SHAPE)
  GENERATE KEY USING GRIDENTRY(SHAPECOL..MBR..XMIN,
                               SHAPECOL..MBR..YMIN,
                               SHAPECOL..MBR..XMAX,
                               SHAPECOL..MBR..YMAX,
                               LEVELS)
  WITH TARGET KEY (LEVEL INT, GX INT, GY INT,
                   XMIN INT, YMIN INT, XMAX INT, YMAX INT)
  SEARCH METHODS
  WHEN SEARCHFIRSTBYSECOND (SEARCHARG SHAPE)
  RANGE THROUGH GRIDRANGE(SEARCHARG..MBR..XMIN,
                               SEARCHARG..MBR..YMIN,
                               SEARCHARG..MBR..XMAX,
                               SEARCHARG..MBR..YMAX,
                               LEVELS)
  FILTER USING
    CASE WHEN (SEARCHARG..MBR..YMIN > YMAX) OR
       SEARCHARG..MBR..YMAX < YMIN) THEN 0
    ELSE CHECKDUPLICATE(LEVEL, GX, GY,
                               XMIN, YMIN, XMAX, YMAX,
                               SEARCHARG..MBR..XMIN,
                               SEARCHARG..MBR..YMIN,
                               SEARCHARG..MBR..XMAX,
                               SEARCHARG..MBR..YMAX,
                               LEVELS)
    END
  WHEN SEARCHSECONDBYFIRST (SEARCHARG SHAPE)
  RANGE THROUGH GRIDRANGE(SEARCHARG..MBR..XMIN,
                               SEARCHARG..MBR..YMIN,
                               SEARCHARG..MBR..XMAX,
                               SEARCHARG..MBR..YMAX,
                               LEVELS)
  FILTER USING
    CASE WHEN (SEARCHARG..MBR..YMIN > YMAX) OR
       SEARCHARG..MBR..YMAX < YMIN) THEN 0
    ELSE MBROVERLAP(XMIN, YMIN, XMAX, YMAX,
                               SEARCHARG..MBR..XMIN,
                               SEARCHARG..MBR..YMIN,
                               SEARCHARG..MBR..XMAX,
                               SEARCHARG..MBR..YMAX)
    END
```

CREATE MASK

The CREATE MASK statement creates a column mask at the current server. A column mask specifies the value to be returned for a specified column.

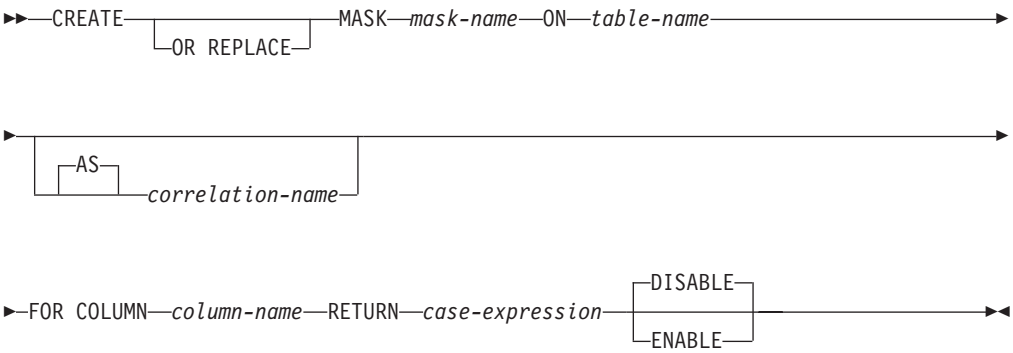
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority. SECADM authority can create a column mask in any schema. Additional privileges are not needed to reference other objects in the mask definition. For example, the SELECT privilege is not needed to retrieve from a table, and the EXECUTE privilege is not needed to call a user-defined function.

Syntax



Description

OR REPLACE
Specifies to replace the definition for the column mask if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog.

mask-name
Names the column mask. The name, including the implicit or explicit qualifier, must not identify a column mask or a row permission that already exists at the current server (SQLSTATE 42710).

table-name
Identifies the table on which the column mask is created. The name must identify a table that exists at the current server (SQLSTATE 42704). It must not identify a nickname, created or declared temporary table, view, synonym, typed table, an alias (SQLSTATE 42809), or a catalog table (SQLSTATE 42832).

correlation-name
Specifies a correlation name that can be used within *case-expression* to designate the table.

FOR COLUMN *column-name*
Identifies the column to which the mask applies. *column-name* must be an

unqualified name that identifies a column of the table (SQLSTATE 42703). A mask must not already exist for the column (SQLSTATE 428HC). The column must not be any of the following columns:

- A LOB column or a distinct type column that is based on a LOB (SQLSTATE 42962).
- An XML column (SQLSTATE 42962).
- A column referenced in an expression that defines a generated column (SQLSTATE 428HB).

RETURN *case-expression*

Specifies a CASE expression to be evaluated to determine the value to return for the column (SQLSTATE 42601). The result of the CASE expression is returned in place of the column value in a row. The result data type, null attribute, and length attribute of the CASE expression must be identical to those of *column-name* (SQLSTATE 428HB). If the data type of *column-name* is a user-defined data type, the result data type of the CASE expression must be the same user-defined data type. The CASE expression must not reference any of the following objects or elements (SQLSTATE 428HB):

- A created global temporary table or a declared global temporary table.
- A nickname.
- A table function.
- A method.
- A parameter marker (SQLSTATE 42601).
- A user-defined function that is defined as not secure.
- A function or expression (such as row change expression, sequence expression) that is non-deterministic or has an external action.
- An XMLQUERY scalar function.
- An XMLEXISTS predicate.
- An OLAP specification.
- A `*` or `name.*` in a SELECT clause.
- A pseudo-column.
- An aggregate function without specifying the SELECT clause.
- A view that includes any of the previously listed restrictions in its definition.

If the CASE expression references tables for which row or column access control is currently activated, access control from those tables are not cascaded. See the Notes section for details.

ENABLE or DISABLE

Specifies that the column mask is to be enabled or disabled for column access control. The default is DISABLE.

DISABLE

Specifies that the column mask is to be disabled for column access control. If column access control is not currently activated for the table, the column mask will remain ineffective when column access control is activated for the table.

ENABLE

Specifies that the column mask is to be enabled for column access control. If column access control is not currently activated for the table, the column mask will become effective when column access control is activated for the table. If column access control is currently activated for the table, the

column mask becomes effective immediately and all packages and dynamic cached statements that reference the table are invalidated.

The application of enabled column masks does not interfere with the operations of other clauses within the statement such as the WHERE, GROUP BY, HAVING, SELECT DISTINCT, and ORDER BY. The rows returned in the final result table remain the same, except that the values in the resulting rows might be masked by the column masks. As such, if the masked column also appears in an ORDER BY *sort-key*, the order is based on the original column values and the masked values in the final result table might not reflect that order. Similarly, the masked values might not reflect the uniqueness enforced by SELECT DISTINCT. If the masked column is embedded in an expression, the result of the expression might become different because the column mask is applied on the column before the expression evaluation can take place. For example, applying a column mask on column SSN might change the result of aggregate function COUNT(DISTINCT SSN) because the DISTINCT operation is performed on the masked values. On the other hand, if the expression in the query is the same as the expression used to mask the column value in the column mask definition, the result of the expression might remain unchanged. For example, the expression in the query is 'XXX-XX-' || SUBSTR(SSN, 8, 4) and the same expression appears in the column mask definition. In this particular example, you can replace the expression in the query with column SSN to avoid the same expression getting evaluated twice.

A column mask is created as a stand alone object without knowing all of the contexts in which it might be used. To mask a column value in the final result table, the column mask definition is merged into a query by the DB2 database. When the column mask definition is brought into the context of the statement, it might conflict with certain SQL semantics in the statement. Therefore, in some situations, the combination of the statement and the application of the column mask might return an error (SQLSTATE 428HD). When this happens, either the statement needs to be modified or the column mask must be dropped or recreated with a different definition. See the ALTER TABLE statement description for those situations where a bind time error might be issued for the statement.

If the column is not nullable, its column mask definition will not consider a null value for the column. After column access control is activated for the target table, if the target table is the null-padded table in an outer join operation, the column value in the final result table might be a null. To ensure the column mask has the ability to mask a null value, when the DB2 database merges the column mask definition into the query, if the target table is the null-padded table in an outer join operation, the DB2 database will add "WHEN *target-column* IS NULL THEN NULL" as the first WHEN clause to the column mask definition. This forces a null value to be always masked to a null. For a nullable column, this takes away the ability to mask a null value to something else but it is an acceptable restriction from security and usability standpoints.

When a column is used to derive the new value for an INSERT, UPDATE, MERGE, or a SET *transition-variable* assignment statement, the original column value, not the masked value, is used to derive the new value. If the column has a column mask, that column mask is applied to ensure the evaluation of the access control rules at run time masks the column to itself, not to a constant or an expression. This is to ensure the masked values are the same as the original column values. If a column mask does not mask the column to itself, the existing row is not updated or the new row is not inserted and an

error is returned at run time (SQLSTATE 428HD). If there is a requirement for masked data to be inserted into a table, it can be done by first assigning the data to a variable. For example, an array variable can be created with the array elements having a row data type. Table data with column masks applied can be assigned to the array variable, which can then be used to insert the data into some other table. The rules that are used to apply column masks in to derive the new values follow the same rules described previously for the final result table of a query. See the INSERT, UPDATE, and MERGE statements for how the column masks are used to affect the insert and update operation.

See the ALTER TABLE statement with the ACTIVATE COLUMN ACCESS CONTROL clause for information about how to activate column access control for the table and how a column mask is applied.

Notes

- **Column masks that are created before column access control is activated for a table:** The CREATE MASK statement is an independent statement that can be used to create a column access control mask before column access control is activated for a table. The only requirement is that the table and the columns exist before the mask is created. Multiple column masks can be created for a table but a column can have one mask only.

  The definition of a mask is stored in the DB2 catalog. Dependency on the table for which the mask is being created and dependencies on other objects referenced in the definition are recorded. No package or dynamic cached statement is invalidated. A column mask can be created as enabled or disabled for column access control. An enabled column mask does not take effect until the ALTER TABLE statement with the ACTIVATE COLUMN ACCESS CONTROL clause is used to activate column access control for the table. SECADM authority is required to issue such an ALTER TABLE statement. A disabled column mask remains ineffective even when column access control is activated for the table. The ALTER MASK statement can be used to alter between ENABLE and DISABLE.

  After column access control is activated for a table, when the table is referenced in a data manipulation statement, all enabled column masks that have been created for the table are implicitly applied by the DB2 database to mask the values returned for the columns referenced in the final result table of the queries or to determine the new values used in the data change statements.

  Creating column masks before activating column access control for a table is the recommended sequence to avoid multiple invalidations of packages and dynamic cached statements that reference the table.
- **Column masks that are created after column access control is activated for a table:** The enabled column masks become effective as soon as they are committed. All the packages and dynamic cached statements that reference the table are invalidated. Thereafter, when the table is referenced in a data manipulation statement, all enabled column masks are implicitly applied by the DB2 database to the statement. Any disabled column masks remain ineffective even when column access control is activated for the table.
- **No cascaded effect when column or row access control enforced tables are referenced in column mask definitions:** A column mask definition can reference tables and columns that are currently enforced by row or column access control. Access control from those tables and columns are ignored when the table for which the column mask is being created is referenced in a data manipulation statement.
- **Consideration for DB2 limits:** If the data manipulation statement already approaches some DB2 limits in the statement, the more enabled column masks

and enabled row permissions are created, the more likely they might affect some limits. This is because the enabled column mask and enabled row permission definitions are implicitly merged into the statement when the table is referenced in a data manipulation statement.

- **Column masks that are enabled but in the invalid state:** If a column mask is enabled for column access control but its state is set to invalid, access to the table on which the column mask is defined is blocked until this situation is resolved (SQLSTATE 560D0).
- **Column masks that return data which is not assignable to the column the mask is defined on:** A column mask can be defined so it can return data which is not assignable to the data type of the column the mask is defined on. When this occurs, the CREATE MASK statement is successful but a cast error will be reported when the mask is applied in a user query.

Examples

- *Example 1*: After column access control is activated for table EMPLOYEE, Paul from the payroll department can see the social security number of the employee whose employee number is 123456. Mary who is a manager can see only the last four characters of the social security number. Peter who is neither role cannot see the social security number.

```
CREATE MASK SSN_MASK ON EMPLOYEE
  FOR COLUMN SSN RETURN
    CASE WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'PAYROLL') = 1)
                                THEN SSN
         WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'MGR') = 1)
                                THEN 'XXX-XX-' || SUBSTR(SSN,8,4)
         ELSE NULL
    END
  ENABLE;
ALTER TABLE EMPLOYEE ACTIVATE COLUMN ACCESS CONTROL;
SELECT SSN FROM EMPLOYEE WHERE EMPNO = 123456;
```

- *Example 2*: In the SELECT statement, column SSN is embedded in an expression that is the same as the expression used in the column mask SSN_MASK. After column access control is activated for table EMPLOYEE, the column mask SSN_MASK is applied to column SSN in the SELECT statement. For this particular expression, the SELECT statement produces the same result as before column access control is activated for all users. The user can replace the expression in the SELECT statement with column SSN to avoid the same expression getting evaluated twice.

```
CREATE MASK SSN_MASK ON EMPLOYEE
  FOR COLUMN SSN RETURN
    CASE WHEN (1 = 1) THEN 'XXX-XX-' || SUBSTR(SSN,8,4)
         ELSE NULL
    END
  ENABLE;
ALTER TABLE EMPLOYEE ACTIVATE COLUMN ACCESS CONTROL;
SELECT 'XXX-XX-' || SUBSTR(SSN,8,4) FROM EMPLOYEE WHERE EMPNO = 123456;
```

- *Example 3*: The California state government conducted a survey for the library usage of the households in each city. Fifty households in each city were sampled in the survey. Each household was given an option, opt-in or opt-out, to show whether their usage in any reports generated from the result of the survey.

  A SELECT statement is used to generate a report to show the average hours used by households in each city. Column mask CITY_MASK is created to mask the city name based on the opt-in or opt-out information chosen by the sampled households. However, after column access control is activated for table LIBRARY_USAGE, the SELECT statement receives a bind time error. This is

because column mask CITY_MASK references another column LIBRARY_OPT and LIBRARY_OPT does not identify a grouping column.

```
CREATE MASK CITY_MASK ON LIBRARY_USAGE
  FOR COLUMN CITY RETURN
    CASE WHEN (LIBRARY_OPT = 'OPT-IN') THEN CITY
         ELSE ' '
    END
  ENABLE;
ALTER TABLE LIBRARY_USAGE ACTIVATE COLUMN ACCESS CONTROL;
SELECT CITY, AVG(LIBRARY_TIME) FROM LIBRARY_USAGE GROUP BY CITY;
```

- *Example 4*: Employee with EMPNO 123456 earns bonus $8000 and salary $80000 in May. When the manager retrieves his salary, the manager receives his salary, not the null value. This is because of no cascaded effect when column mask SALARY_MASK references column BONUS for which column mask BONUS_MASK is defined.

```
CREATE MASK SALARY_MASK ON EMPLOYEE
  FOR COLUMN SALARY RETURN
    CASE WHEN (BONUS < 10000) THEN SALARY
         ELSE NULL
    END
  ENABLE;
CREATE MASK BONUS_MASK ON EMPLOYEE
  FOR COLUMN BONUS RETURN
    CASE WHEN (BONUS > 5000) THEN NULL
         ELSE BONUS
    END
  ENABLE;
ALTER TABLE EMPLOYEE ACTIVATE COLUMN ACCESS CONTROL;
SELECT SALARY FROM EMPLOYEE WHERE EMPNO = 123456;
```

CREATE METHOD

The CREATE METHOD statement is used to associate a method body with a method specification that is already part of the definition of a user-defined structured type.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CREATEIN privilege on the schema of the structured type referred to in the CREATE METHOD statement
- The owner of the structured type referred to in the CREATE METHOD statement
- DBADM authority

To associate an external method body with its method specification, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

- CREATE_EXTERNAL_ROUTINE authority on the database
- DBADM authority

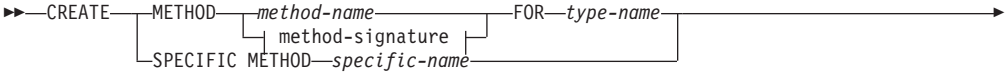
When creating an SQL method, the privileges held by the authorization ID of the statement must also include at least one of the following authorities for each table, view, or nickname identified in any fullselect:

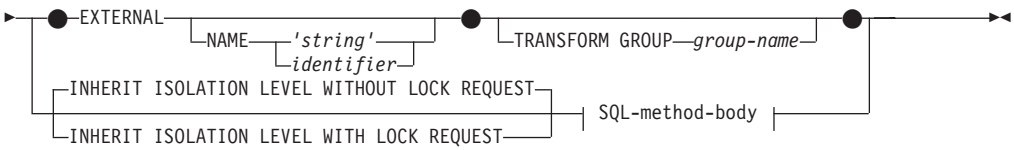
- CONTROL privilege on that table, view, or nickname
- SELECT privilege on that table, view, or nickname
- DATAACCESS authority

Group privileges other than PUBLIC are not considered for any table or view specified in the CREATE METHOD statement.

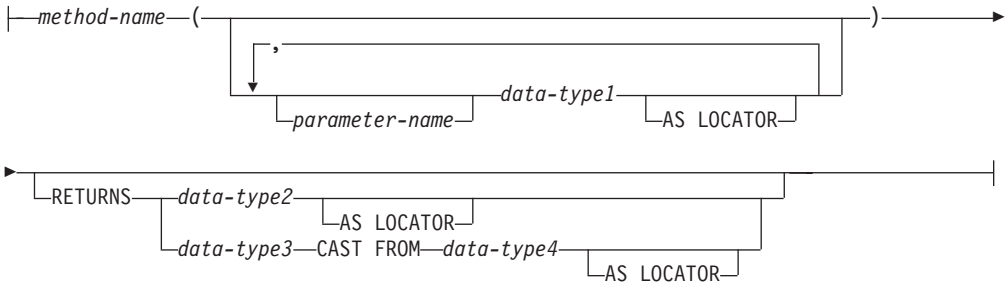
Authorization requirements of the data source for the table or view referenced by the nickname are applied when the method is invoked. The authorization ID of the connection can be mapped to a different remote authorization ID.

Syntax





method-signature:



SQL-method-body:

Notes:

1 The compound SQL (inlined) statement is only supported for an SQL-method-body in an SQL method definition in a non-partitioned database.

Description

METHOD

Identifies an existing method specification that is associated with a user-defined structured type. The method-specification can be identified through one of the following means:

method-name

Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*). There must be only one method specification for *type-name* that has this *method-name* (SQLSTATE 42725).

method-signature

Provides the method signature which uniquely identifies the method to be defined. The method signature must match the method specification that was provided on the CREATE TYPE or ALTER TYPE statement (SQLSTATE 42883).

method-name

Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*).

parameter-name

Identifies the parameter name. If parameter names are provided in the method signature, they must be exactly the same as the

corresponding parts of the matching method specification. Parameter names are supported in this statement solely for documentation purposes.

data-type1
Specifies the data type of each parameter. Array types are not supported (SQLSTATE 42815).

For a more complete description of each built-in data type, see "CREATE TABLE".

AS LOCATOR
For the LOB types or distinct types which are based on a LOB type, the AS LOCATOR clause can be added.

RETURNS
This clause identifies the output of the method. If a RETURNS clause is provided in the method signature, it must be exactly the same as the corresponding part of the matching method specification on CREATE TYPE. The RETURNS clause is supported in this statement solely for documentation purposes.

data-type2
Specifies the data type of the output. Array types are not supported (SQLSTATE 42815).

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be returned by the method instead of the actual value.

data-type3 **CAST FROM** *data-type4*
This form of the RETURNS clause is used to return a different data type to the invoking statement from the data type that was returned by the function code.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be used to indicate that a LOB locator is to be returned from the method instead of the actual value.

FOR *type-name*
Names the type for which the specified method is to be associated. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

SPECIFIC METHOD *specific-name*
Identifies the particular method, using the specific name either specified or defaulted to at CREATE TYPE time. The specific-name must identify a method specification in the named or implicit schema; otherwise, an error is raised (SQLSTATE 42704).

EXTERNAL
This clause indicates that the CREATE METHOD statement is being used to register a method, based on code written in an external programming language, and adhering to the documented linkage conventions and interface.

The matching method-specification in CREATE TYPE must specify a LANGUAGE other than SQL. When the method is invoked, the subject of the method is passed to the implementation as an implicit first parameter.

If the NAME clause is not specified, "NAME *method-name*" is assumed.

NAME
This clause identifies the name of the user-written code which implements the method being defined.

'string'
The 'string' option is a string constant with a maximum of 254 bytes. The format used for the string is dependent on the LANGUAGE specified. For more information about the specific language conventions, see “CREATE FUNCTION (External Scalar) statement”.

identifier
This identifier specified is an SQL identifier. The SQL identifier is used as the library-id in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C (as defined in the method-specification on CREATE TYPE).

TRANSFORM GROUP *group-name*
Indicates the transform group that is used for user-defined structured type transformations when invoking the method. A transform is required since the method definition includes a user-defined structured type.

It is strongly recommended that a transform group name be specified; if this clause is not specified, the default group-name used is DB2_FUNCTION. If the specified (or default) group-name is not defined for a referenced structured type, an error results (SQLSTATE 42741). Likewise, if a required FROM SQL or TO SQL transform function is not defined for the given group-name and structured type, an error results (SQLSTATE 42744).

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST or INHERIT ISOLATION LEVEL WITH LOCK REQUEST
Specifies whether or not a lock request can be associated with the isolation-clause of the statement when the method inherits the isolation level of the statement that invokes the method. The default is INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST.

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
Specifies that, as the method inherits the isolation level of the invoking statement, it cannot be invoked in the context of an SQL statement which includes a lock-request-clause as part of a specified isolation-clause (SQLSTATE 42601).

INHERIT ISOLATION LEVEL WITH LOCK REQUEST
Specifies that, as the method inherits the isolation level of the invoking statement, it also inherits the specified lock-request-clause.

SQL-method-body
The SQL-method-body defines how the method is implemented if the method specification in CREATE TYPE is LANGUAGE SQL.

The SQL-method-body must comply with the following parts of method specification:

- DETERMINISTIC or NOT DETERMINISTIC (SQLSTATE 428C2)
- EXTERNAL ACTION or NO EXTERNAL ACTION (SQLSTATE 428C2)

- CONTAINS SQL or READS SQL DATA (SQLSTATE 42985)

Parameter names can be referenced in the SQL-method-body. The subject of the method is passed to the method implementation as an implicit first parameter named SELF.

For additional details, see "Compound SQL (inlined) statement" and "RETURN statement".

Rules

- The method specification must be previously defined using the CREATE TYPE or ALTER TYPE statement before CREATE METHOD can be used (SQLSTATE 42723).
- If the method being created is an overriding method, those packages that are dependent on the following methods are invalidated:
  - The original method
  - Other overriding methods that have as their subject a supertype of the method being created
- The XML data type cannot be used in a method.

Notes

- If the method allows SQL, the external program must not attempt to access any federated objects (SQLSTATE 55047).
- ***Privileges***: The definer of a method always receives the EXECUTE privilege on the method, as well as the right to drop the method.

  If an EXTERNAL method is created, the definer of the method always receives the EXECUTE privilege WITH GRANT OPTION.

  If an SQL method is created, the definer of the method will only be given the EXECUTE privilege WITH GRANT OPTION on the method when the definer has WITH GRANT OPTION on all privileges required to define the method, or if the definer has SYSADM or DBADM authority. The definer of an SQL method only acquires privileges if the privileges from which they are derived exist at the time the method is created. The definer must have these privileges either directly, or because PUBLIC has the privileges. Privileges held by groups of which the method definer is a member are not considered. When using the method, the connected user's authorization ID must have the valid privileges on the table or view that the nickname references at the data source.
- ***Table access restrictions***: If a method is defined as READS SQL DATA, no statement in the method can access a table that is being modified by the statement which invoked the method (SQLSTATE 57053).

Examples

- *Example 1:*

```
CREATE METHOD BONUS (RATE DOUBLE)
  FOR EMP
  RETURN SELF..SALARY * RATE
```

- *Example 2:*

```
CREATE METHOD SAMEZIP (addr address_t)
  RETURNS INTEGER
  FOR address_t
  RETURN
    (CASE
```

```
      WHEN (self..zip = addr..zip)
        THEN 1
      ELSE 0
    END)
```

- *Example 3:*

```
CREATE METHOD DISTANCE (address_t)
  FOR address_t
  EXTERNAL NAME 'addresslib!distance'
  TRANSFORM GROUP func_group
```

CREATE MODULE

The CREATE MODULE statement creates a module at the application server.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

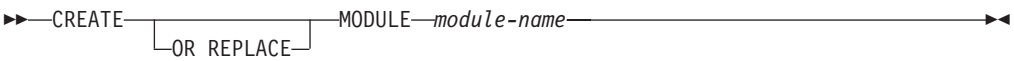
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the module does not exist.
- CREATEIN privilege on the schema, if the schema name of the module refers to an existing schema.
- DBADM authority

To replace an existing module, the authorization ID of the statement must be the owner of the existing module (SQLSTATE 42501).

Syntax



Description

OR REPLACE
Specifies replacing the definition for the module if one exists at the current server. The existing module definition is effectively dropped, including all the objects in the module, before the new definition is replaced in the catalog, with the exception that privileges that were granted on the module are not affected. This option is ignored if a definition for the module does not exist at the current server. This option can be specified only by the owner of the object.

module-name
Names the module. The name, including the implicit or explicit qualifier, must not identify an existing module at the current server. The module name and the schema name must not begin with the characters 'SYS' (SQLSTATE 42939) and use of SESSION is not recommended.

Notes

- A module is intended to be a collection of other database objects. Once a module is created, objects in the module are managed using the ALTER MODULE statement. A module can include functions, procedures, types, global variables and conditions. The objects in a module can be published to make them available for reference from outside the module. If an object is not published, it can only be referenced from within the module. A module can be considered to consist of 2 parts:
  - The module specification consists of all the published objects excluding the bodies of any routines.

– The module body which consists of all objects that are not published and the bodies of any published routines.

The module management actions include

– ADD to add an object to the module without publishing it or to replace a routine prototype with the implemented routine definition.
– PUBLISH to add an object to the module and publish it.
– COMMENT on objects in the module.
– DROP to drop an object within the module or drop the module body.

At least one published object should exist in a module in order to have some way to reference the module.

Example

Create a module named *salesModule*

```
CREATE MODULE salesModule
```

CREATE NICKNAME

The CREATE NICKNAME statement defines a nickname for a data source object.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

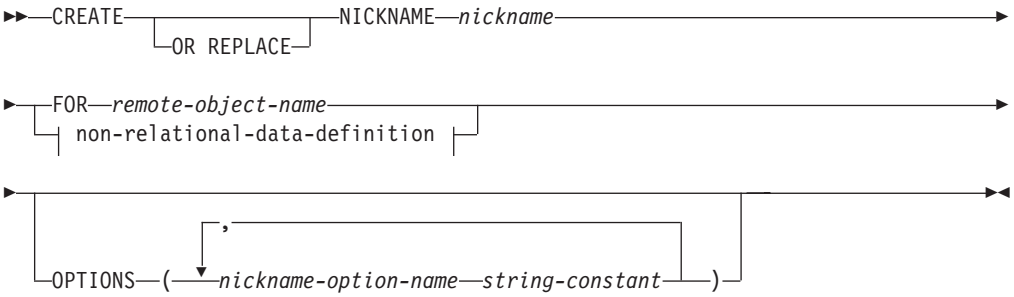
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CREATETAB authority on the federated database, as well as one of:
  - IMPLICIT_SCHEMA authority on the federated database, if the implicit or explicit schema name of the nickname does not exist
  - CREATEIN privilege on the schema, if the schema name of the nickname refers to an existing schema
- DBADM authority

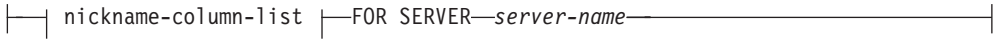
For data sources that require a user mapping, the privileges held by the authorization ID at the data source must include the privilege to select data from the object that the nickname represents.

To replace an existing nickname, the authorization ID of the statement must be the owner of the existing nickname (SQLSTATE 42501).

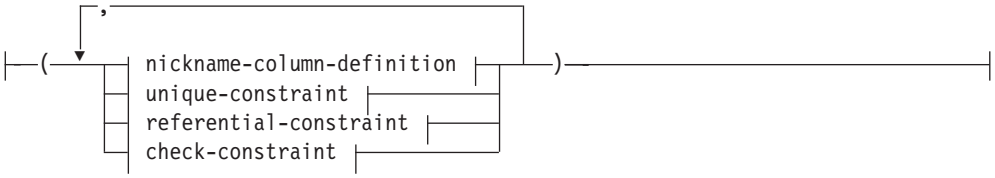
Syntax



non-relational-data-definition:



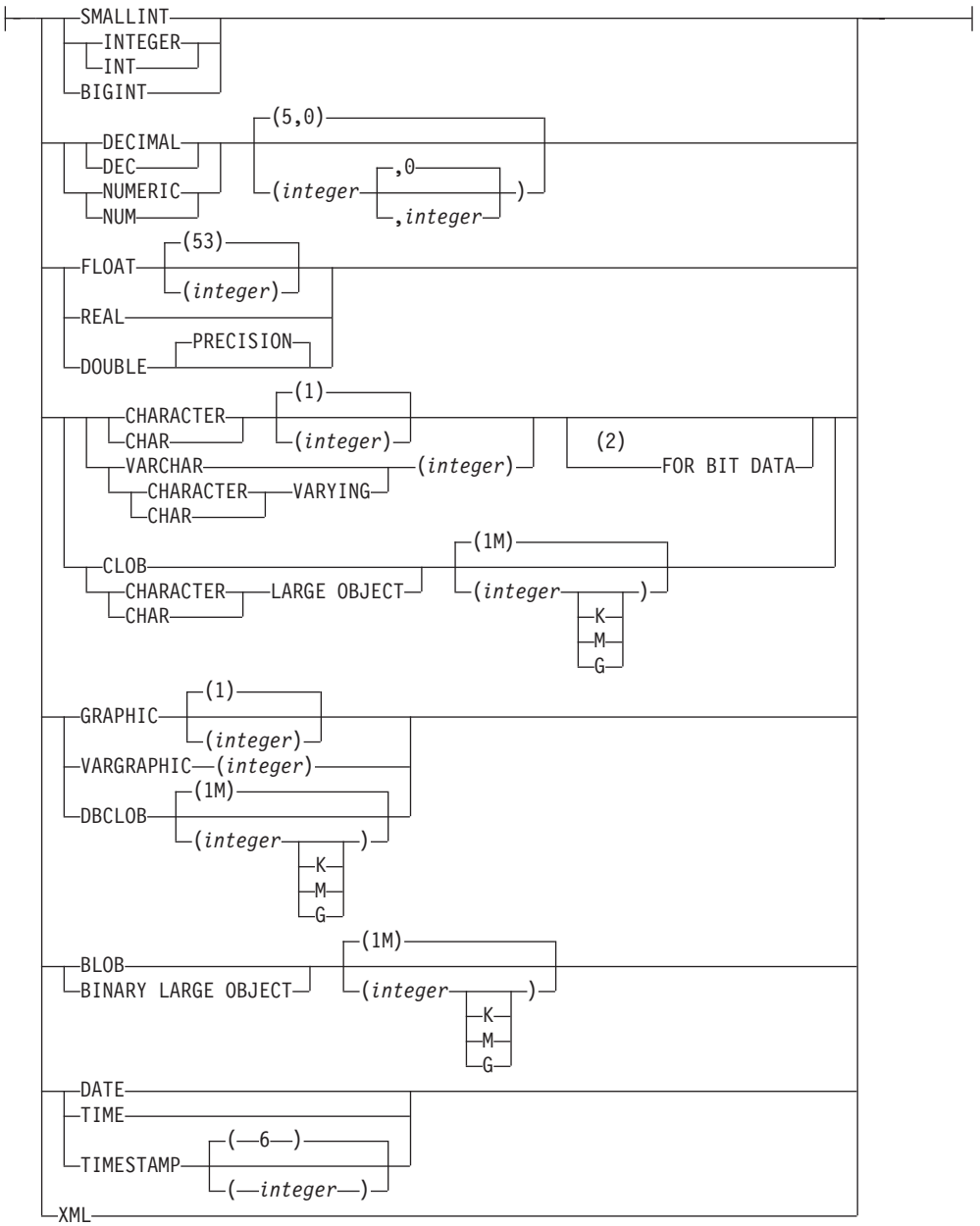
nickname-column-list:



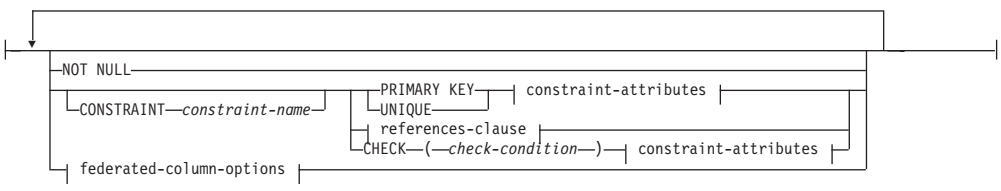
nickname-column-definition:

local-data-type:

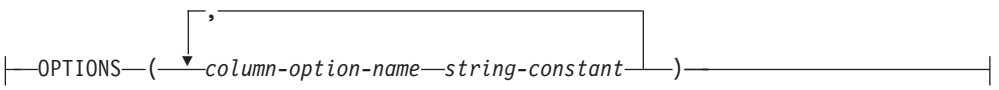
built-in-type:



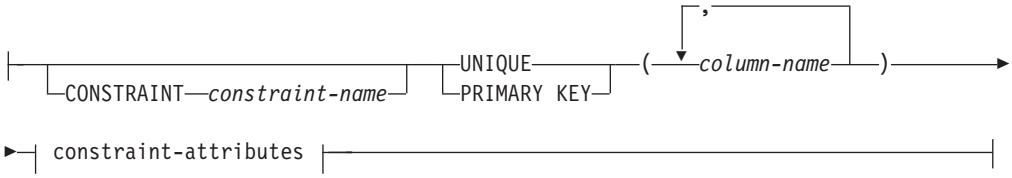
nickname-column-options:



federated-column-options:

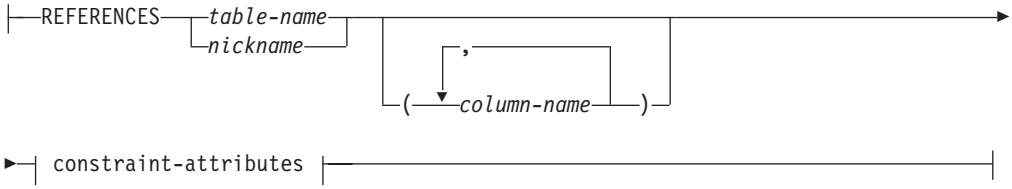


unique-constraint:

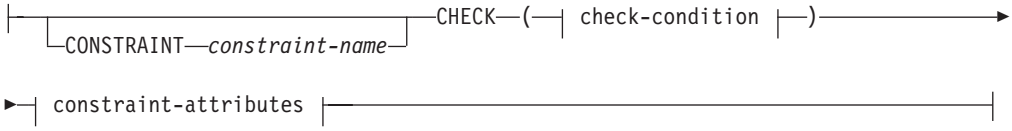


referential-constraint:

references-clause:

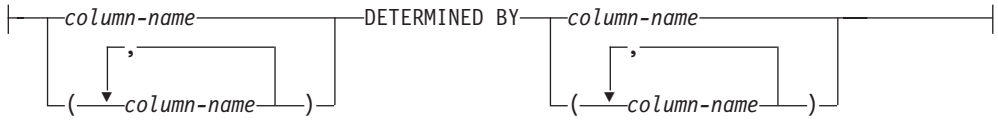


check-constraint:

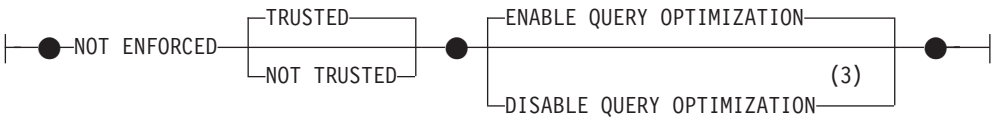


check-condition:

functional-dependency:



constraint-attributes:



Notes:

1 The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type (SQLSTATE 428H2).

2 The FOR BIT DATA clause can be specified in any order with the other column constraints that follow.

3 DISABLE QUERY OPTIMIZATION is not supported for a unique or primary key constraint.

Description

OR REPLACE
Specifies to replace the definition for the nickname if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the nickname are not affected. This option is ignored if a definition for the nickname does not exist at the current server. This option can be specified only by the owner of the object.

nickname
Specifies a nickname, the identifier used by the federated server for the data source object. The nickname, including the implicit or explicit qualifier, must not identify a table, view, nickname, or alias described in the catalog. The schema name must not begin with 'SYS' (SQLSTATE 42939).

FOR *remote-object-name*
Specifies an identifier. For data sources that support schema names, this is a three-part identifier with the format *data-source-name.remote-schema-name.remote-table-name*. For data sources that do not support schema names, this is a two-part identifier with the format *data-source-name.remote-table-name*.

data-source-name
Names the data source that contains the table or view for which the nickname is being created. The *data-source-name* is the same name that was assigned to the *server-name* in the CREATE SERVER statement.

remote-schema-name
Names the schema to which the table or view belongs. If the remote schema name contains any special or lowercase characters, it must be enclosed by double quotation marks.

remote-table-name
Names the specific data source object (such as a table, alias of a table, or view) for which the nickname is being created. The table cannot be a declared temporary table (SQLSTATE 42995). If the remote table name contains any special or lowercase characters, it must be enclosed by double quotation marks. For DB2 for Linux, UNIX, and Windows you can also specify the alias of a table, view, or nickname. For DB2 for z/OS or DB2 for i, you can specify the alias of a table or view.

non-relational-data-definition
Defines the data that is to be accessed through a nonrelational wrapper.

nickname-column-definition
Defines the local attributes of the column for the nickname. Some wrappers require these attributes to be specified, while other wrappers allow the attributes to be determined from the data source.

column-name
Specifies the local name for the column. The name might be different than the corresponding column of the *remote-object-name*.

local-data-type
Specifies the local data type for the column. Some wrappers only support a subset of the SQL data types. For descriptions of specific data types, see "CREATE TABLE" .

nickname-column-options
Specifies additional options related to columns of the nickname.

NOT NULL
Specifies that the column does not allow null values.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same CREATE NICKNAME statement (SQLSTATE 42710).

If this clause is omitted, an 18 byte long identifier that is unique among the identifiers of existing constraints defined on the nickname is generated by the system. (The identifier consists of 'SQL' followed by a sequence of 15 numeric characters generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* can be used as the name of an index specification that is created to support the constraint.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

See PRIMARY KEY within the description of *unique-constraint*.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

See UNIQUE within the description of *unique-constraint*.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* within the description of *referential-constraint*.

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See description for CHECK (*check-condition*).

OPTIONS
Indicates the column options that are added when the nickname is created. Some wrappers require that certain column options be specified.

column-option-name
Specifies the name of the option.

string-constant
Specifies the setting for *column-option-name* as a character string constant.

unique-constraint
Defines a unique or primary key constraint.

CONSTRAINT *constraint-name*
Names the primary key or unique constraint.

UNIQUE (*column-name***,...)**
Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the nickname and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL and XQuery limits". No LOB column, distinct type column based on a LOB, or structured type column can be used as part of a unique key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the unique key cannot be the same as the set of columns in the primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

The description of the nickname as recorded in the catalog includes the unique key and its index specification. An index specification will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index specification will be the same as the *constraint-name* if this does not conflict with an existing index or index specification in the schema where the nickname is created. If the name of the index specification conflicts, the name will be 'SQL' followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

PRIMARY KEY (*column-name***,...)**
Defines a primary key composed of the identified columns. The clause must not be specified more than once, and the identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the nickname, and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE

TABLE". For key length limits, see "SQL and XQuery limits". No LOB column, distinct type column based on a LOB, or structured type column can be used as part of a primary key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

Only one primary key can be defined on a nickname.

The description of the nickname as recorded in the catalog includes the primary key and its index specification. An index specification will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index specification will be the same as the *constraint-name* if this does not conflict with an existing index or index specification in the schema where the nickname is created. If the name of the index specification conflicts, the name will be 'SQL', followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

referential-constraint

Defines a referential constraint.

CONSTRAINT *constraint-name*

Names the referential constraint.

FOREIGN KEY (*column-name***,...)**

Defines a referential constraint with the specified *constraint-name*.

Let N1 denote the object nickname of the statement. The foreign key of the referential constraint is composed of the identified columns. Each name in the list of column names must identify a column of N1, and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL and XQuery limits". Foreign keys can be defined on variable length columns whose length is greater than 255 bytes. No LOB column, distinct type column based on a LOB, or structured type column can be used as part of a foreign key (SQLSTATE 42962). There must be the same number of foreign key columns as there are in the parent key, and the data types of the corresponding columns must be compatible (SQLSTATE 42830). Two column descriptions are compatible if they have compatible data types (both columns are numeric, character string, graphic, datetime, or have the same distinct type).

references-clause

Specifies the parent table or the parent nickname, and the parent key for the referential constraint.

REFERENCES *table-name* **or** *nickname*

The table or nickname specified in a REFERENCES clause must identify a base table or a nickname that is described in the catalog, but must not identify a catalog table.

A referential constraint is a duplicate if its foreign key, parent key, and parent table or parent nickname are the same as the foreign key, parent key, and parent table or parent nickname of a previously specified

referential constraint. Duplicate referential constraints are ignored, and a warning is returned (SQLSTATE 01543).

In the following discussion, let N2 denote the identified parent table or parent nickname, and let N1 denote the nickname being created (or altered). N1 and N2 may be the same nickname.

The specified foreign key must have the same number of columns as the parent key of N2, and the description of the *n*th column of the foreign key must be comparable to the description of the *n*th column of that parent key. Datetime columns are not considered to be comparable to string columns for the purposes of this rule.

The referential constraint specified by a FOREIGN KEY clause defines a relationship in which N2 is the parent and N1 is the dependent.

(*column-name***,...)**
The parent key of a referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of N2. The same column must not be identified more than once.

The list of column names must match the set of columns (in any order) of the primary key or a unique constraint that exists on N2 (SQLSTATE 42890). If a column name list is not specified, N2 must have a primary key (SQLSTATE 42888). Omission of the column name list is an implicit specification of the columns of that primary key in the sequence originally specified.

constraint-attributes
Defines attributes associated with referential integrity or check constraints.

NOT ENFORCED
The constraint is not enforced by the database manager during normal operations, such as insert, update, or delete.

TRUSTED
The data can be trusted to conform to the constraint. TRUSTED must be used only if the data in the table is independently known to conform to the constraint. Query results might be unpredictable if the data does not actually conform to the constraint. This is the default option.

NOT TRUSTED
The data cannot be trusted to conform to the constraint. NOT TRUSTED is intended for cases where the data conforms to the constraint for most rows, but it is not independently known that all the rows or future additions will conform to the constraint. If a constraint is NOT TRUSTED and enabled for query optimization, then it will not be used to perform optimizations that depend on the data conforming completely to the constraint. NOT TRUSTED can be specified only for referential integrity constraints (SQLSTATE 42613).

ENABLE QUERY OPTIMIZATION
The constraint is assumed to be true and can be used for query optimization under appropriate circumstances.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

check-constraint

Defines a check constraint. A *check-constraint* is a *search-condition* that must evaluate to not false or that defines a functional dependency between columns.

CONSTRAINT *constraint-name*

Names the check constraint.

CHECK (*check-condition***)**

Defines a check constraint. The *check-condition* must be true or unknown for every row of the nickname.

search-condition

The *search-condition* has the following restrictions:

- A column reference must be to a column of the nickname being created.
- The *search-condition* cannot contain a TYPE predicate.
- It cannot contain any of the following elements (SQLSTATE 42621):
  - Subqueries
  - Dereference operations or DEREF functions where the scoped reference argument is other than the object identifier (OID) column
  - CAST specifications with a SCOPE clause
  - Column functions
  - Functions that are not deterministic
  - Functions defined to have an external action
  - User-defined functions defined with either CONTAINS SQL or READS SQL DATA
  - Host variables
  - Parameter markers
  - Special registers and built-in functions that depend on the value of a special register
  - Global variables
  - References to generated columns other than the identity column

functional-dependency

Defines a functional dependency between columns.

The parent set of columns contains the identified columns that immediately precede the DETERMINED BY clause. The child set of columns contains the identified columns that immediately follow the DETERMINED BY clause. All of the restrictions on the *search-condition* apply to parent set and child set columns, and only simple column references are allowed in the set of columns (SQLSTATE 42621). The same column must not be identified more than once in the functional dependency (SQLSTATE 42709). The data type of the column must not be a LOB data type, a distinct type based on a LOB data type, or a structured type (SQLSTATE 42962). No column in the child set of columns can be a nullable column (SQLSTATE 42621).

If a check constraint is specified as part of a *column-definition*, a column reference can only be made to the same column. Check constraints specified as part of a nickname definition can have column references identifying columns previously defined in the CREATE NICKNAME statement. Check constraints are not checked for inconsistencies, duplicate

conditions, or equivalent conditions. Therefore, contradictory or redundant check constraints can be defined, resulting in possible errors at execution time.

FOR SERVER *server-name*
Specifies a server that was registered using the CREATE SERVER statement. This server will be used to access the data for the nickname.

OPTIONS
Indicates the nickname options that are enabled when the nickname is created.

nickname-option-name
Specifies the name of the option.

string-constant
Specifies the setting for *nickname-option-name* as a character string constant.

Notes

- Examples of relational data source objects are: tables and views. Examples of nonrelational data source objects are: Documentum objects or registered tables, text files (.txt), and Microsoft Excel files (.xls).
- The data source object that the nickname references must already exist at the data source denoted by the first qualifier in *remote-object-name*.
- The list of supported data source data types varies from wrapper to wrapper. XML and REF data source data types are not supported by any of the wrappers. DECFLOAT data source data type is supported only by the DB2 wrapper for IBM DB2 for Linux, UNIX, and Windows Version 9.5 or later. When the CREATE NICKNAME statement specifies a *remote-object-name* that has columns with unsupported data types, an error is returned.

  LONG VARCHAR and LONG VARGRAPHIC data source data types are mapped to CLOB and DBCLOB data types, respectively. LONG VARCHAR FOR BIT DATA is mapped to BLOB.
- The maximum allowable length of DB2 index names is 128 bytes. If a nickname is being created for a relational table that has an index whose name exceeds this length, the entire name is not cataloged. Rather, DB2 truncates it to 128 bytes. If the string formed by these characters is not unique within the schema to which the index belongs, DB2 attempts to make it unique by replacing the last character with 0. If the result is still not unique, DB2 changes the last character to 1. DB2 repeats this process with numbers 2 through 9 and, if necessary, with numbers 0 through 9 for the name's 127th character, 126th character, and so on, until a unique name is generated. To illustrate: The 130-byte name of an index on a data source table is AREALLY...REALLYLONGNAME. The names AREALLY...REALLYLONGNA and AREALLY...REALLYLONGN0 already exist in the schema to which this index belongs. The new name is over 128 bytes; therefore, DB2 truncates it to AREALLY...REALLYLONGNA. Because this name already exists in the schema, DB2 changes the truncated version to AREALLY...REALLYLONGN0. And because this name also exists, DB2 changes the truncated version to AREALLY...REALLYLONGN1. This name does not already exist in the schema, so DB2 accepts it as a new name.
- When a nickname is created for a data source object, DB2 stores the names of the nickname columns in the catalog. When the data source object is a table or a view, DB2 makes the nickname column names the same as the table or view column names. If a name exceeds the maximum allowable length for DB2 column names, DB2 truncates the name to this length. If the truncated version is not unique among the other column names in the table or view, DB2 makes it unique by following the procedure described in the preceding paragraph.

- If the data source object has indexes defined, index specifications for each index are created when the nickname is created. Index specifications are not created at the data source for indexes that have:
  - Duplicate column names
  - More than 64 columns
  - More than 1024 bytes in the sum of the length of the index key parts
- If the definition of a remote data source object is changed (for example, a column is deleted or a data type is changed), the nickname should be dropped and recreated; otherwise, errors might occur when the nickname is used in an SQL statement.
- ***Caching and protected objects:*** When a nickname is created, if the data source object is not protected, `ALLOW CACHING` is in effect for the nickname. If the federated server can detect that the data source object is protected, `DISALLOW CACHING` is in effect for the nickname. The `DISALLOW CACHING` option ensures that each time the nickname is used, data for the appropriate authorization ID is returned from the data source at query execution time. This is done by restricting the nickname from being used in the definition of a materialized query table at the federated server, which might be being used to cache the nickname data. The ALTER NICKNAME statement can be used to change between `ALLOW CACHING` and `DISALLOW CACHING`.
- **Syntax alternatives:** The following syntax is supported for compatibility with previous versions of DB2:
  - ADD can be specified before *nickname-option-name string-constant*.
  - ADD can be specified before *column-option-name string-constant*.

Examples

- *Example 1:* Create a nickname for a view, DEPARTMENT, that is in a schema called HEDGES. This view is stored in a DB2 for z/OS data source called OS390A.

```
CREATE NICKNAME DEPT
  FOR OS390A.HEDGES.DEPARTMENT
```

- *Example 2:* Select all records from the view for which a nickname was created in Example 1. The view must be referenced by its nickname. The remote view can be referenced using the name by which it is known at the data source only in pass-through sessions.

  The following statement is valid after nickname DEPT is created:

```
SELECT * FROM DEPT
```

  The following statement is invalid:

```
SELECT * FROM OS390A.HEDGES.DEPARTMENT
```

- *Example 3:* Create a nickname for the remote table JAPAN that is in a schema called salesdata. Because the schema name and table name on the data source are stored in lowercase, specify the remote schema name and table name with double quotation marks:

```
CREATE NICKNAME JPSALES
  FOR asia."salesdata"."japan"
```

- *Example 4:* Create a nickname for the table-structured file DRUGDATA1.TXT. Include the FILE_PATH, COLUMN DELIMITER, KEY_COLUMN, and VALIDATE_DATA_FILE nickname options in the statement.

```
CREATE NICKNAME DRUGDATA1
  (Dcode         INTEGER,
  DRUG           CHAR(20),
  MANUFACTURER   CHAR(20))
  FOR SERVER biochem_lab
```

```
OPTIONS
  (FILE_PATH '/usr/pat/DRUGDATA1.TXT',
   COLUMN_DELIMITER ',',
   KEY_COLUMN 'DCODE',
   SORTED 'Y',
   VALIDATE_DATA_FILE 'Y')
```

- *Example 5:* Create the parent nickname CUSTOMERS over multiple XML files under the specified directory path /home/db2user. Include the following options:
  - Column options:
    - XPATH column option for the VARCHAR(5) column named ID, indicating the element or attribute in the XML file(s) from which the column data is extracted
    - XPATH column option for the VARCHAR(16) column named NAME, indicating the element or attribute in the XML file(s) from which the column data is extracted
    - XPATH column option for the VARCHAR(30) column named ADDRESS, indicating the element or attribute in the XML file(s) from which the column data is extracted
    - PRIMARY_KEY column option for the VARCHAR(16) column named CID, which identifies the customers nickname as a parent nickname in a hierarchy of nicknames
  - Nickname options:
    - DIRECTORY_PATH nickname option to indicate the location of the XML files that provide the data
    - XPATH nickname option to indicate the element in the XML files where the data begins
    - STREAMING nickname option to indicate that the XML source data is separated and processed element by element. In this example, the element is a customer record.

```
CREATE NICKNAME customers
  (id      VARCHAR(5)   OPTIONS(XPATH './@id'),
   name    VARCHAR(16)  OPTIONS(XPATH './/name'),
   address VARCHAR(30)  OPTIONS(XPATH './/address/@street'),
   cid     VARCHAR(16)  OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS
     (DIRECTORY_PATH '/home/db2user',
      XPATH '//customer',
      STREAMING 'YES')
```

CREATE PERMISSION

The CREATE PERMISSION statement creates a row permission at the current server.

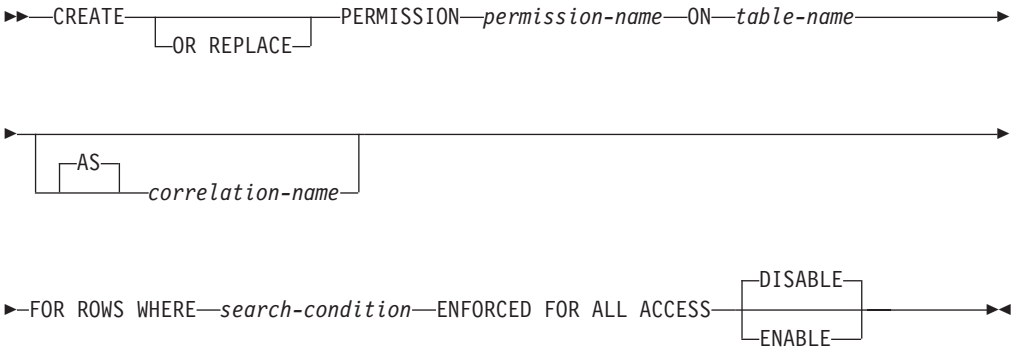
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority. SECADM authority can create a row permission in any schema. Additional privileges are not needed to reference other objects in the permission definition. For example, the SELECT privilege is not needed to retrieve from a table, and the EXECUTE privilege is not needed to call a user-defined function.

Syntax



Description

OR REPLACE
Specifies to replace the definition for the row permission if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog.

permission-name
Names the row permission. The name, including the implicit or explicit qualifier, must not identify a row permission or a column mask that already exists at the current server (SQLSTATE 42710).

table-name
Identifies the table on which the row permission is created. The name must identify a table that exists at the current server (SQLSTATE 42704). It must not identify a nickname, created or declared temporary table, view, alias, synonym, or a typed table (SQLSTATE 42809). Also, it must not identify a catalog table (SQLSTATE 42832).

correlation-name
Specifies a correlation name that can be used within *case-expression* to designate the table.

FOR ROWS WHERE
Indicates that a row permission is created. A row permission specifies a search condition under which rows of the table can be accessed.

search-condition
Specifies a condition that can be true or false for a row of the table. This follows the same rules used by the search condition in a WHERE clause of a subselect query. In addition, the search condition must not reference any of the following objects or elements (SQLSTATE 428HB):

- A created global temporary table or a declared global temporary table.
- A nickname.
- A table function.
- A method.
- A parameter marker (SQLSTATE 42601).
- A user-defined function that is defined as not secure.
- A function or expression (such as row change expression, sequence expression) that is non deterministic or has an external action
- An XMLQUERY scalar function.
- An XMLEXISTS predicate.
- An OLAP specification.
- A `*` or `name.*` in a SELECT clause.
- A pseudocolumn.
- An aggregate function without specifying the SELECT clause.
- A view that includes any of the previously listed restrictions in its definition.

If *search-condition* references tables with currently activated row or column access control, access control from those tables are not cascaded. See "Notes" for details.

ENFORCED FOR ALL ACCESS
Specifies that the row permission applies to all references of the table. If row access control is activated for the table, when the table is referenced in a data manipulation statement, the DB2 database implicitly applies the row permission to control the access of the table. If the reference of the table is for a fetch operation such as SELECT, the application of the row permission determines what set of rows can be retrieved by the user who requested the fetch operation. If the reference of the table is for a data change operation such as INSERT, the application of the row permission determines whether all rows to be changed can be inserted or updated by the user who requested the data change operation.

ENABLE or DISABLE
Specifies that the row permission is to be enabled or disabled. The default is DISABLE.

DISABLE
Specifies that the row permission is to be disabled. If row access control is not currently activated for the table, the row permission will remain ineffective when row access control is activated for the table.

ENABLE
Specifies that the row permission is to be enabled for row access control. If row access control is not currently activated for the table, the row permission will become effective when row access control is activated for

the table. If row access control is currently activated for the table, the row permission becomes effective immediately and all packages and dynamically cached statements that reference the table are invalidated.

See the ACTIVATE ROW ACCESS CONTROL clause in the ALTER TABLE statement for more information about how to activate row access control and how row permissions are applied.

Notes

- **Row permissions that are created before row access control is activated for a table:** The CREATE PERMISSION statement is an independent statement that can be used to create a row permission before row access control is activated for a table. The only requirement is that the table and the columns exist before the permission is created. Multiple row permissions can be created for a table.

  The definition of the row permission is stored in the DB2 catalog. Dependency on the table for which the permission is being created and dependencies on other objects referenced in the definition are recorded. No package or dynamic cached statement is invalidated. A row permission can be created as enabled or disabled for row access control. An enabled row permission does not take effect until the ALTER TABLE statement with the ACTIVATE ROW ACCESS CONTROL clause is used to activate row access control for the table. A disabled row permission remains ineffective even when row access control is activated for the table. The ALTER PERMISSION statement can be used to alter between ENABLE and DISABLE.

  After row access control is activated for a table, when the table is referenced in a data manipulation statement, all enabled row permissions that are defined for the table are implicitly applied by the DB2 database to control access to the table.

  Creating row permissions before activating row access control for a table is the recommended sequence to avoid multiple invalidations of packages and dynamic cached statements that reference the table.
- **Row permissions that are created after row access control is activated for a table:** An enabled row permission becomes effective as soon as it is committed. All the packages and dynamic cached statements that reference the table are invalidated. Thereafter, when the table is referenced in a data manipulation statement, all enabled row permissions are implicitly applied to the statement. Any disabled row permissions remain ineffective even when row access control is activated for the table.
- **No cascaded effect when row or column access control enforced tables are referenced in row permission definitions:** A row permission definition might reference tables and columns that are currently enforced by row or column access control. Access control from those tables are ignored when the table for which the row permission is being created is referenced in a data manipulation statement.
- **Consideration for DB2 limits:** If the data manipulation statement already approaches some DB2 limits in the statement, the more enabled row permissions and enabled column masks are created, the more likely they might affect some limits. This is because the enabled column mask and enabled row permission definitions are implicitly merged into the statement when the table is referenced in a data manipulation statement. See "SQL and XML Limits" for the limits of a statement.

- **Permissions that are enabled but in the invalid state:** If a permission is enabled for row access control but its state is set to invalid, access to the table on which the permission is defined is blocked until this situation is resolved (SQLSTATE 560D0).

Example

The tellers in a bank can only access customers from their own branch. All tellers are members in role TELLER. The customer service representatives are allowed to access all customers of the bank. All customer service representatives are members in role CSR. A row permission is created accordingly for each group of personnel in the bank by a user with SECADM authority. After row level access control is activated for table CUSTOMER, in the SELECT statement the search conditions of both row permissions are merged into the statement and they are combined with the logical OR operator to control the set of rows accessible by each group.

```
CREATE PERMISSION TELLER_ROW_ACCESS ON CUSTOMER
 FOR ROWS WHERE VERIFY_ROLE_FOR_USER
 (SESSION_USER,'TELLER') = 1 AND
        BRANCH = (SELECT HOME_BRANCH FROM INTERNAL_INFO
             WHERE EMP_ID = SESSION_USER)
ENFORCED FOR ALL ACCESS
ENABLE;

CREATE PERMISSION CSR_ROW_ACCESS ON CUSTOMER
 FOR ROWS WHERE VERIFY_ROLE_FOR_USER(SESSION_USER,'CSR') = 1
 ENFORCED FOR ALL ACCESS
 ENABLE;
```

CREATE PROCEDURE

The CREATE PROCEDURE statement defines a procedure at the current server.

Three different types of procedures can be created using this statement. Each of these types is described separately.

- External. The procedure body is written in a programming language. The external executable is referenced by a procedure defined at the current server, along with various attributes of the procedure.
- Sourced. The procedure body is part of the source procedure, which is referenced by the sourced procedure that is defined at the current server, along with various attributes of the procedure. A sourced procedure whose source procedure is at a data source is also called a *federated procedure*.
- SQL. The procedure body is written in SQL and defined at the current server, along with various attributes of the procedure.

The CREATE PROCEDURE statement can be submitted in obfuscated form. In an obfuscated statement, only the procedure name and its parameters are readable. The rest of the statement is encoded in such a way that is not readable but can be decoded by the database server. Obfuscated statements can be produced by calling the DBMS_DDL.WRAP function.

CREATE PROCEDURE (external)

The CREATE PROCEDURE (external) statement defines an external procedure at the current server.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CREATE_EXTERNAL_ROUTINE authority on the database and at least one of the following authorities:
  - IMPLICIT_SCHEMA authority on the database, if the schema name of the procedure does not refer to an existing schema
  - CREATEIN privilege on the schema, if the schema name of the procedure refers to an existing schema
- DBADM authority

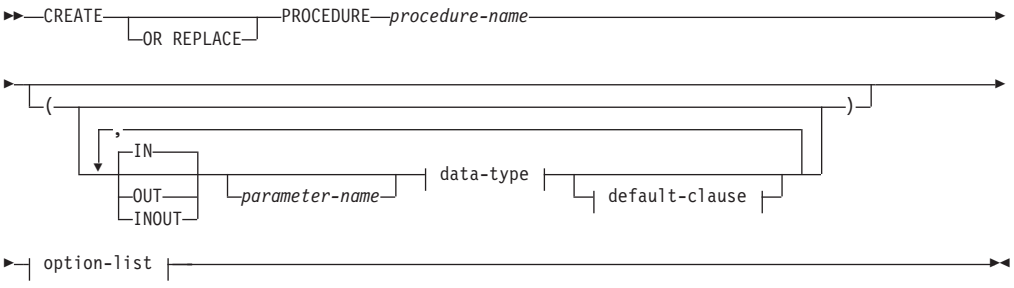
To create a not-fenced procedure, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

- CREATE_NOT_FENCED_ROUTINE authority on the database
- DBADM authority

To create a fenced procedure, no additional authorities or privileges are required.

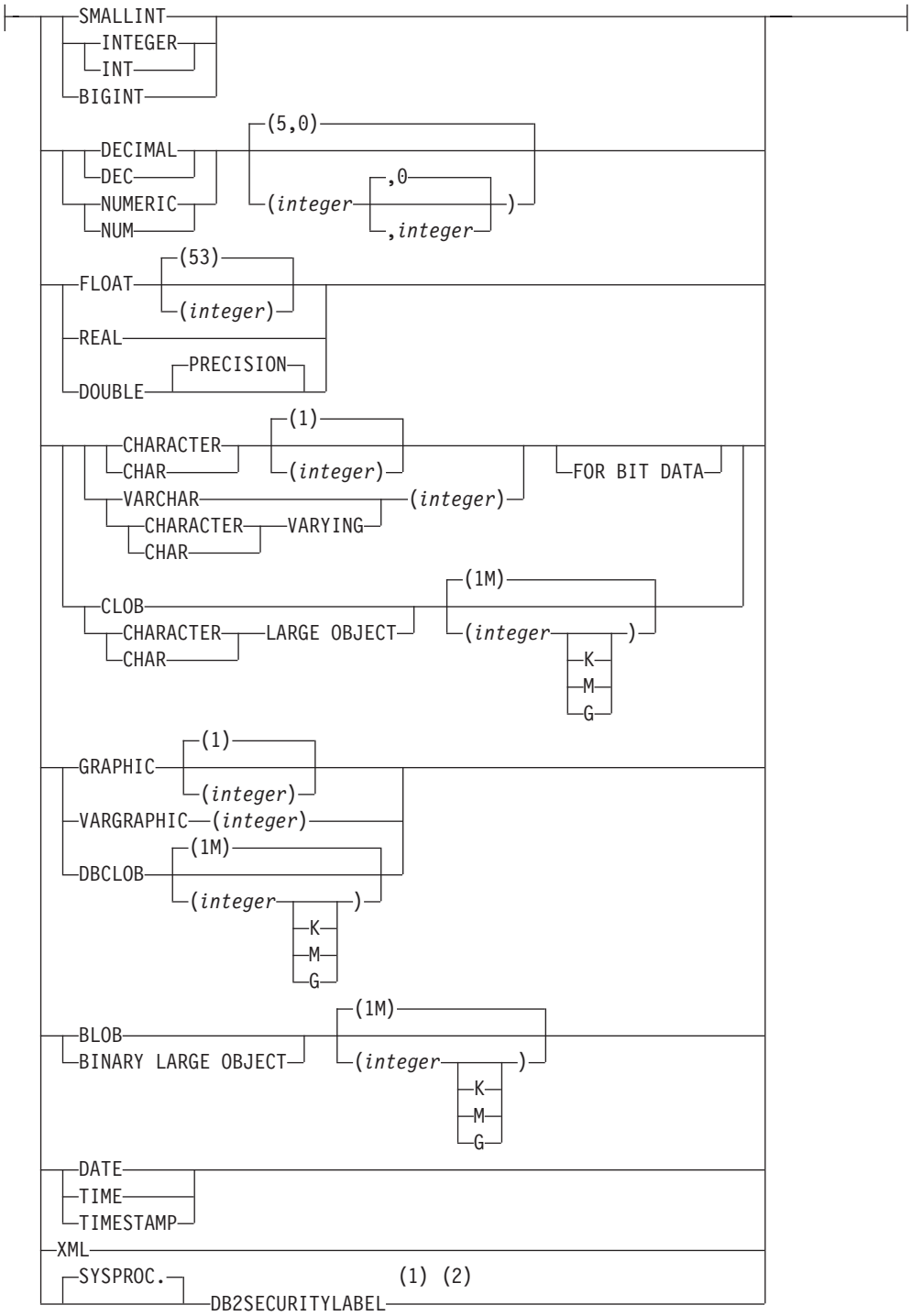
To replace an existing procedure, the authorization ID of the statement must be the owner of the existing procedure (SQLSTATE 42501).

Syntax



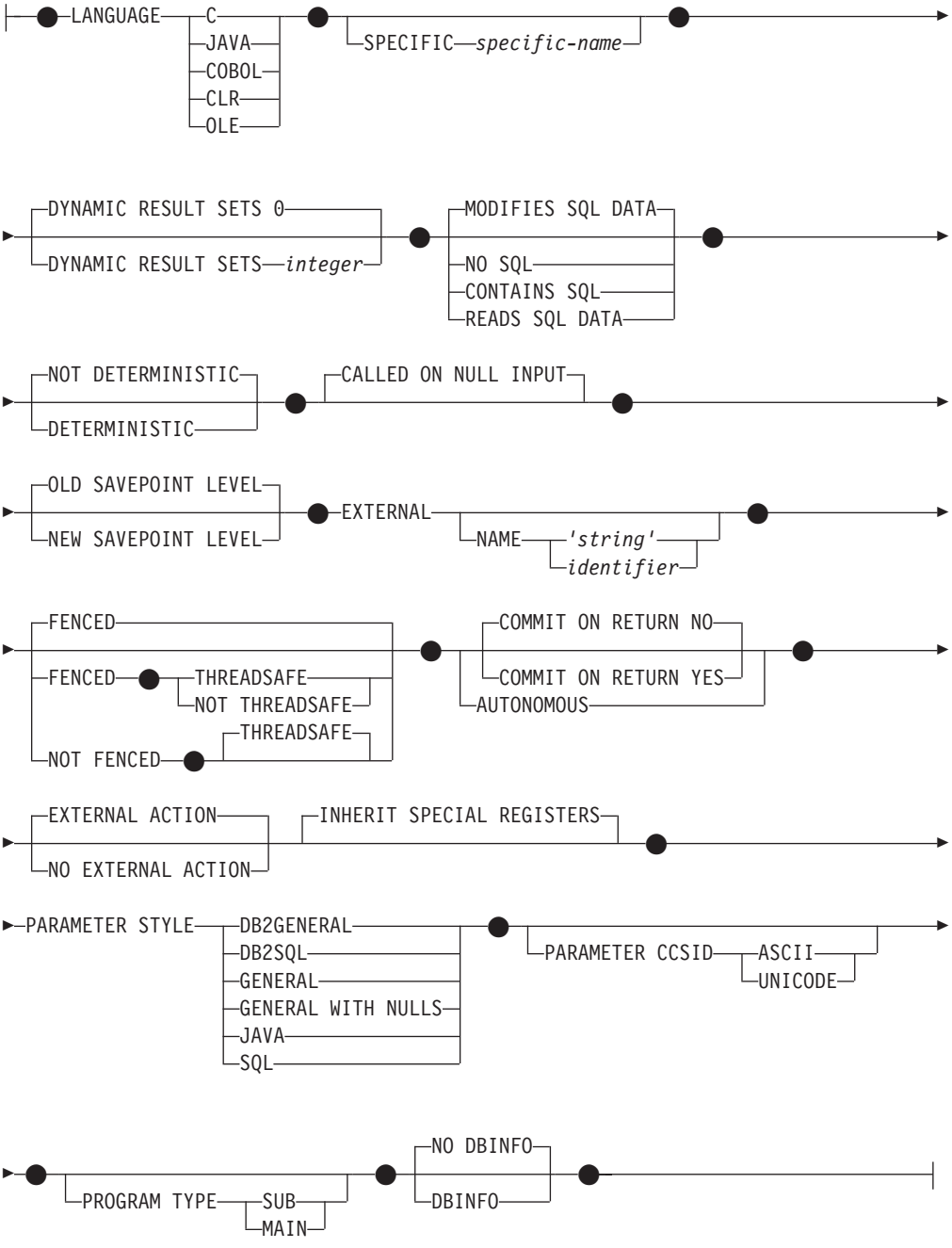
data-type:

built-in-type:



default-clause:

option-list:



Notes:

1 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

2 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.

Description

OR REPLACE
Specifies to replace the definition for the procedure if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the procedure are not affected. This option can be specified only by the owner of the object. This option is ignored if a definition for the procedure does not exist at the current server. To replace an existing procedure, the specific name and procedure name of the new definition must be the same as the specific name and procedure name of the old definition, or the signature of the new definition must match the signature of the old definition. Otherwise, a new procedure is created.

procedure-name
Names the procedure being defined. It is a qualified or unqualified name that designates a procedure. The unqualified form of *procedure-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters must not identify a procedure described in the catalog (SQLSTATE 42723). The unqualified name, together with the number of the parameters, need not be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

*(***IN** | **OUT** | **INOUT** *parameter-name data-type default-clause,...)*
Identifies the parameters of the procedure, and specifies the mode, optional parameter name, data type, and optional default value of each parameter. One entry in the list must be specified for each parameter that the procedure will expect.

No two identically-named procedures within a schema are permitted to have exactly the same number of parameters. A duplicate signature returns an SQL error (SQLSTATE 42723).

For example, given the statements:

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART_NAME CHAR(35)) ...
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

the second statement will fail, because the number of parameters in the procedure is the same, even if the data types are not.

If an error is returned by the procedure, OUT parameters are undefined and INOUT parameters are unchanged.

IN Identifies the parameter as an input parameter to the procedure. Any changes made to the parameter within the procedure are not available to the calling SQL application when control is returned. The default is IN.

OUT
Identifies the parameter as an output parameter for the procedure.

INOUT
Identifies the parameter as both an input and output parameter for the procedure.

parameter-name
Optionally specifies the name of the parameter. The parameter name must be unique for the procedure (SQLSTATE 42734).

data-type
Specifies the data type of the parameter. A structured type cannot be specified (SQLSTATE 429BB).

built-in-type
Specifies a built-in data type. For a more complete description of each built-in data type, see "CREATE TABLE". Only built-in data types that have a correspondence in the language that is being used to write the procedure may be specified.

- A datetime type parameter is passed as a character data type, and the data is passed in the ISO format.
- XML is invalid with LANGUAGE OLE.
- Because the XML value that is seen inside a procedure is a serialized version of the XML value that is passed as a parameter in the procedure call, parameters of type XML must be declared using the syntax `XML AS CLOB(n)`.
- CLR does not support DECIMAL scale greater than 28 (SQLSTATE 42613).
- Decimal floating-point is not supported with languages C, Java COBOL, CLR, and OLE (SQLSTATE 42613).

array-type-name
Specifies the name of a user-defined array type. If *array-type-name* is specified without a schema name, the array type is resolved by searching the schemas in the SQL path. The array must be an ordinary array and the procedure must be a Java procedure defined with the PARAMETER STYLE JAVA clause (SQLSTATE 428H2).

DEFAULT
Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see "*default-clause*" in the "CREATE TABLE" statement). Other special registers can be specified as the default by using an expression.

The *expression* can be any expression of the type described in "Expressions". If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified in the following situations:

- For INOUT or OUT parameters (SQLSTATE 42601)
- For a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB)

SPECIFIC *specific-name*
Provides a unique name for the instance of the procedure that is being defined. This specific name can be used when altering, dropping, or commenting on the procedure. It can never be used to invoke the procedure. The unqualified form of *specific-name* is an SQL identifier. The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another routine instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* may be the same as an existing *procedure-name*.

If no qualifier is specified, the qualifier that was used for *procedure-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *procedure-name* or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is 'SQL' followed by a character timestamp: 'SQL*yymmddhhmmssxxx*'.

DYNAMIC RESULT SETS *integer*
Indicates the estimated upper bound of returned result sets for the procedure.

NO SQL, CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA
Indicates whether the procedure issues any SQL statements and, if so, what type.

NO SQL
Indicates that the procedure cannot execute any SQL statements (SQLSTATE 38001).

CONTAINS SQL
Indicates that SQL statements that neither read nor modify SQL data can be executed by the procedure (SQLSTATE 38004). Statements that are not supported in any procedure return a different error (SQLSTATE 38003).

READS SQL DATA
Indicates that some SQL statements that do not modify SQL data can be included in the procedure (SQLSTATE 38002 or 42985). Statements that are not supported in any procedure return a different error (SQLSTATE 38003).

MODIFIES SQL DATA
Indicates that the procedure can execute any SQL statement except statements that are not supported in procedures (SQLSTATE 38003).

DETERMINISTIC or NOT DETERMINISTIC
This clause specifies whether the procedure always returns the same results for given argument values (DETERMINISTIC) or whether the procedure depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC procedure must always return the same result from successive invocations with identical inputs.

This clause currently does not impact processing of the procedure.

CALLED ON NULL INPUT
CALLED ON NULL INPUT always applies to procedures. This means that the procedure is called regardless of whether any arguments are null. Any OUT or INOUT parameter can return a null value or a normal (non-null) value. Responsibility for testing for null argument values lies with the procedure.

OLD SAVEPOINT LEVEL or NEW SAVEPOINT LEVEL
Specifies whether or not this procedure establishes a new savepoint level for savepoint names and effects. OLD SAVEPOINT LEVEL is the default behavior. For more information about savepoint levels, see the "Rules" section in the description of the SAVEPOINT statement.

LANGUAGE
This mandatory clause is used to specify the language interface convention to which the procedure body is written.

C This means the database manager will call the procedure as if it were a C procedure. The procedure must conform to the C language calling and linkage convention as defined by the standard ANSI C prototype.

JAVA
This means the database manager will call the procedure as a method in a Java class.

COBOL
This means the database manager will call the procedure as if it were a COBOL procedure.

CLR
This means the database manager will call the procedure as a method in a .NET class. At this time, LANGUAGE CLR is only supported for procedures running on Windows operating systems. NOT FENCED cannot be specified for a CLR routine (SQLSTATE 42601).

OLE
This means the database manager will call the procedure as if it were a method exposed by an OLE automation object. The stored-procedure must conform with the OLE automation data types and invocation mechanism. Also, the OLE automation object needs to be implemented as an in-process server (DLL). These restrictions are outlined in the *OLE Automation Programmer's Reference*.

LANGUAGE OLE is only supported for procedures stored in DB2 for Windows operating systems. THREADSAFE may not be specified for procedures defined with LANGUAGE OLE (SQLSTATE 42613).

EXTERNAL
This clause indicates that the CREATE PROCEDURE statement is being used to register a new procedure based on code written in an external programming language and adhering to the documented linkage conventions and interface.

If the NAME clause is not specified, "NAME *procedure-name*" is assumed. If the NAME clause is not formatted correctly, an error is returned (SQLSTATE 42878).

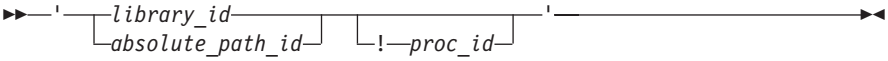
NAME *'string'*
This clause identifies the name of the user-written code which implements the procedure being defined.

The 'string' option is a string constant with a maximum of 254 bytes. The format used for the string is dependent on the LANGUAGE specified.

- For LANGUAGE C:

  The *string* specified is the library name and procedure within the library, which the database manager invokes to execute the procedure being CREATEd. The library (and the procedure within the library) do not need to exist when the CREATE PROCEDURE statement is performed.

However, when the procedure is called, the library and procedure within the library must exist and be accessible from the database server machine.



The name must be enclosed by single quotation marks. Extraneous blanks are not permitted.

library_id
Identifies the library name containing the procedure. The database manager will look for the library as follows:

- On UNIX systems, if 'myfunc' was given as the *library_id*, and the database manager is being run from /u/production, the database manager will look for the procedure in library /u/production/sqllib/function/myproc if FENCED is specified, or /u/production/sqllib/function/unfenced/myproc if NOT FENCED is specified.
- On Windows operating systems, the database manager will look for the function in a directory path that is specified by the LIBPATH or PATH environment variable.

Stored procedures located in any of these directories do not use any of the registered attributes.

absolute_path_id
Identifies the full path name of the procedure.

On UNIX systems, for example, '/u/jchui/mylib/myproc' would cause the database manager to look in /u/jchui/mylib for the myproc procedure.

On Windows operating systems, 'd:\mylib\myproc.dll' would cause the database manager to load the file myproc.dll from the d:\mylib directory. If an absolute path ID is being used to identify the routine body, be sure to append the .dll extension.

! proc_id
Identifies the entry point name of the procedure to be invoked. The exclamation point (!) serves as a delimiter between the library ID and the procedure ID. '!proc8' would direct the database manager to look for the library in the location specified by *absolute_path_id*, and to use entry point proc8 within that library.

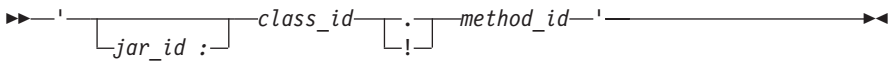
If the string is not properly formed, an error is returned (SQLSTATE 42878).

The body of every procedure should be in a directory that is mounted and available on every database partition.

- For LANGUAGE JAVA:

  The *string* specified contains the optional jar file identifier, class identifier and method identifier, which the database manager invokes to execute the procedure being CREATEd. The class identifier and method identifier do not need to exist when the CREATE PROCEDURE statement is performed. If a *jar_id* is specified, it must exist when the CREATE PROCEDURE statement is performed. However, when the

procedure is called, the class identifier and the method identifier must exist and be accessible from the database server machine, otherwise an error is returned (SQLSTATE 42884).



The name must be enclosed by single quotation marks. Extraneous blanks are not permitted.

jar_id
Identifies the jar identifier given to the jar collection when it was installed in the database. It can be either a simple identifier or a schema qualified identifier. Examples are 'myJar' and 'mySchema.myJar'.

class_id
Identifies the class identifier of the Java object. If the class is part of a package, the class identifier part must include the complete package prefix, for example, 'myPacks.StoredProcs'. The Java virtual machine will look in directory '../myPacks/StoredProcs/' for the classes. In Windows operating systems, the Java virtual machine will look in directory '..\myPacks\StoredProcs\'.

method_id
Identifies the method name with the Java class to be invoked.

- For LANGUAGE CLR:

  The *string* specified represents the .NET assembly (library or executable), the class within that assembly, and the method within the class that the database manager invokes to execute the procedure being created. The module, class, and method do not need to exist when the CREATE PROCEDURE statement is executed. However, when the procedure is called, the module, class, and method must exist and be accessible from the database server machine, otherwise an error is returned (SQLSTATE 42284).

  C++ routines that are compiled with the '/clr' compiler option to indicate that they include managed code extensions must be cataloged as 'LANGUAGE CLR' and not 'LANGUAGE C'. DB2 needs to know that the .NET infrastructure is being utilized in a procedure in order to make necessary runtime decisions. All procedures using the .NET infrastructure must be cataloged as 'LANGUAGE CLR'.

The name must be enclosed by single quotation marks. Extraneous blanks are not permitted.

assembly
Identifies the DLL or other assembly file in which the class resides. Any file extensions (such as .dll) must be specified. If the full path name is not given, the file must reside in the function directory of the DB2 instance path (for example, c:\DB2\function). If the file resides in a subdirectory of the instance function directory, the subdirectory can be given before the file name rather than specifying the full path. For example, if your instance directory is c:\DB2 and your assembly file is c:\DB2\function\myprocs\mydotnet.dll, it is

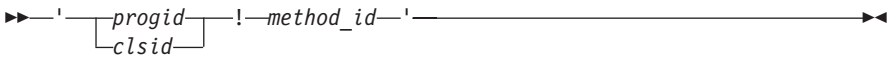
only necessary to specify 'myprocs\mydotnet.dll' for the assembly. The case sensitivity of this parameter is the same as the case sensitivity of the file system.

class_id
Specifies the name of the class within the given assembly in which the method that is to be invoked resides. If the class resides within a namespace, the full namespace must be given in addition to the class. For example, if the class EmployeeClass is in namespace MyCompany.ProcedureClasses, then MyCompany.ProcedureClasses.EmployeeClass must be specified for the class. Note that the compilers for some .NET languages will add the project name as a namespace for the class, and the behavior may differ depending on whether the command line compiler or the GUI compiler is used. This parameter is case sensitive.

method_id
Specifies the method within the given class that is to be invoked. This parameter is case sensitive.

- For LANGUAGE OLE:

  The string specified is the OLE programmatic identifier (*progid*) or class identifier (*clsid*), and method identifier (*method_id*), which the database manager invokes to execute the procedure being created by the statement. The programmatic identifier or class identifier, and the method identifier do not need to exist when the CREATE PROCEDURE statement is executed. However, when the procedure is used in the CALL statement, the method identifier must exist and be accessible from the database server machine, otherwise an error results (SQLSTATE 42724).



The name must be enclosed by single quotation marks. Extraneous blanks are not permitted.

progid
Identifies the programmatic identifier of the OLE object.

A *progid* is not interpreted by the database manager, but only forwarded to the OLE automation controller at run time. The specified OLE object must be creatable and support late binding (also known as IDispatch-based binding). By convention, progids have the following format:

```
<program_name>.<component_name>.<version>
```

Because this is only a convention, and not a rule, *progids* may in fact have a different format.

clsid
Identifies the class identifier of the OLE object to create. It can be used as an alternative for specifying a progid in the case that an OLE object is not registered with a progid. The *clsid* has the form:

```
{nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn}
```

where 'n' is an alphanumeric character. A *clsid* is not interpreted by the database manager, but only forwarded to the OLE APIs at run time.

method_id
Identifies the method name of the OLE object to be invoked.

NAME *identifier*
This *identifier* specified is an SQL identifier. The SQL identifier is used as the *library-id* in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C.

FENCED or NOT FENCED
This clause specifies whether the procedure is considered "safe" to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED).

If a procedure is registered as FENCED, the database manager protects its internal resources (for example, data buffers) from access by the procedure. All procedures have the option of running as FENCED or NOT FENCED. In general, a procedure running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for procedures that have not been adequately checked out can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that could occur, but cannot guarantee complete integrity when NOT FENCED procedures are used.

Either SYSADM authority, DBADM authority, or a special authority (CREATE_NOT_FENCED) is required to register a procedure as NOT FENCED. Only FENCED can be specified for a procedure with LANGUAGE OLE or NOT THREADSAFE.

LANGUAGE CLR procedures cannot be created when specifying the NOT FENCED clause (SQLSTATE 42601).

THREADSAFE or NOT THREADSAFE
Specifies whether the procedure is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the procedure is defined with LANGUAGE other than OLE:

- If the procedure is defined as THREADSAFE, the database manager can invoke the procedure in the same process as other routines. In general, to be threadsafe, a procedure should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED procedures can be THREADSAFE.
- If the procedure is defined as NOT THREADSAFE, the database manager will never invoke the procedure in the same process as another routine.

For FENCED procedures, THREADSAFE is the default if the LANGUAGE is JAVA or CLR. For all other languages, NOT THREADSAFE is the default. If the procedure is defined with LANGUAGE OLE, THREADSAFE may not be specified (SQLSTATE 42613).

For NOT FENCED procedures, THREADSAFE is the default. NOT THREADSAFE cannot be specified (SQLSTATE 42613).

COMMIT ON RETURN
Indicates whether a commit is to be issued on return from the procedure. The default is NO.

NO A commit is not issued when the procedure returns.

YES
A commit is issued when the procedure returns if a positive SQLCODE is returned by the CALL statement

The commit operation includes the work that is performed by the calling application process and the procedure.

If the procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

AUTONOMOUS
Indicates the procedure should execute in its own autonomous transaction scope.

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not (NO EXTERNAL ACTION). The default is EXTERNAL ACTION. If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

INHERIT SPECIAL REGISTERS
This optional clause specifies that updatable special registers in the procedure will inherit their initial values from the environment of the invoking statement.

No changes to the special registers are passed back to the caller of the procedure.

Non-updatable special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore set to their default values.

PARAMETER STYLE
This clause is used to specify the conventions used for passing parameters to and returning the value from procedures.

DB2GENERAL
This means that the procedure will use a parameter passing convention that is defined for use with Java methods. This can only be specified when LANGUAGE JAVA is used.

DB2SQL
In addition to the parameters on the CALL statement, the following arguments are passed to the procedure:

- A vector containing a null indicator for each parameter on the CALL statement
- The SQLSTATE to be returned to DB2
- The qualified name of the procedure
- The specific name of the procedure
- The SQL diagnostic string to be returned to DB2

This can only be specified when LANGUAGE C, COBOL, CLR, or OLE is used.

GENERAL
This means that the procedure will use a parameter passing mechanism by which the procedure receives the parameters specified on the CALL. The

parameters are passed directly, as expected by the language; the SQLDA structure is not used. This can only be specified when LANGUAGE C, COBOL, or CLR is used.

Null indicators are *not* directly passed to the program.

GENERAL WITH NULLS
In addition to the parameters on the CALL statement specified under GENERAL, another argument is passed to the procedure. This additional argument is a vector of null indicators, one for each of the parameters on the CALL statement. In C, this would be an array of short integers. This can only be specified when LANGUAGE C, COBOL, or CLR is used.

JAVA
This means that the procedure will use a parameter passing convention that conforms to the Java language and SQLJ Routines specification. IN/OUT and OUT parameters will be passed as single entry arrays to facilitate returning values. This can only be specified when LANGUAGE JAVA is used.

PARAMETER STYLE JAVA procedures do not support the DBINFO or PROGRAM TYPE clauses.

SQL
In addition to the parameters on the CALL statement, the following arguments are passed to the procedure:

- A null indicator for each parameter on the CALL statement
- The SQLSTATE to be returned to DB2
- The qualified name of the procedure
- The specific name of the procedure
- The SQL diagnostic string to be returned to DB2

This can only be specified when LANGUAGE C, COBOL, CLR, or OLE is used.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the procedure. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031). When the procedure is invoked, the application code page for the procedure is the database code page.

UNICODE
Specifies that string data is encoded in Unicode. If the database is a Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8. In either case, when the procedure is invoked, the application code page for the procedure is 1208.

If the database is not a Unicode database, and a procedure with PARAMETER CCSID UNICODE is created, the procedure cannot have any graphic types, the XML type, or user-defined types (SQLSTATE 560C1). PARAMETER CCSID UNICODE procedures can only be called from a DB2 Version 8.1 or later client (SQLSTATE 42997).

If the database is not a Unicode database, and the alternate collating sequence has been specified in the database configuration, procedures can be created with either PARAMETER CCSID ASCII or PARAMETER CCSID UNICODE. All data passed into and out of the procedure will be converted to the appropriate code page.

This clause cannot be specified with LANGUAGE OLE, LANGUAGE JAVA, or LANGUAGE CLR (SQLSTATE 42613).

PROGRAM TYPE
Specifies whether the procedure expects parameters in the style of a main routine or a subroutine. The default is SUB.

SUB
The procedure expects the parameters to be passed as separate arguments.

MAIN
The procedure expects the parameters to be passed as an argument counter, and a vector of arguments (argc, argv). The name of the procedure to be invoked must also be "main". Stored procedures of this type must still be built in the same fashion as a shared library, rather than a stand-alone executable. PROGRAM TYPE MAIN is only valid when the LANGUAGE clause specifies one of: C, COBOL, or CLR.

DBINFO or NO DBINFO
Specifies whether specific information known by DB2 is passed to the procedure when it is invoked as an additional invocation-time argument (DBINFO) or not (NO DBINFO). NO DBINFO is the default. DBINFO is not supported for LANGUAGE OLE (SQLSTATE 42613). It is also not supported for PARAMETER STYLE JAVA or DB2GENERAL.

If DBINFO is specified, a structure containing the following information is passed to the procedure:

- Data base name - the name of the currently connected database.
- Application ID - unique application ID which is established for each connection to the database.
- Application Authorization ID - the authorization ID of the user that connected to the database (the SYSTEM_USER special register).
- Code page - identifies the database code page.
- Database version/release - identifies the version, release and modification level of the database server invoking the procedure.
- Platform - contains the server's platform type.

The DBINFO structure is common for all external routines and contains additional fields that are not relevant to procedures.

If you change session authorization ID (the SESSION_USER special register) using the SET SESSION AUTHORIZATION statement, the Application Authorization ID still returns the value of the SYSTEM_USER special register.

Rules

- ***Autonomous routine restrictions:*** Autonomous routines cannot return result sets and do not support the following parameter data types (SQLSTATE 428H2):
  - Cursor types
  - Structured types
  - XML

Global variables of cursor types cannot be referenced within the autonomous scope.

Notes

- Creating a procedure with a schema name that does not already exist results in the implicit creation of that schema, provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- A Java routine defined as NOT FENCED will be invoked as if it had been defined as FENCED THREADSAFE.
- A procedure that is called from within a compound SQL (inlined) statement will execute as if it were created specifying NEW SAVEPOINT LEVEL, even if OLD SAVEPOINT LEVEL was specified or defaulted to when the procedure was created.
- XML parameters are only supported in LANGUAGE JAVA external procedures when the PARAMETER STYLE DB2GENERAL clause is specified.
- ***Setting of the default value:*** Parameters of a procedure that are defined with a default value are set to their default value when the procedure is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the procedure is invoked.
- ***Privileges:*** The definer of a procedure always receives the EXECUTE privilege WITH GRANT OPTION on the procedure, as well as the right to drop the procedure. When the procedure is used in an SQL statement, the procedure definer must have the EXECUTE privilege on any packages used by the procedure.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - RESULT SETS can be specified in place of DYNAMIC RESULT SETS.
  - NULL CALL can be specified in place of CALLED ON NULL INPUT.
  - DB2GENRL can be specified in place of DB2GENERAL.
  - SIMPLE CALL can be specified in place of GENERAL.
  - SIMPLE CALL WITH NULLS can be specified in place of GENERAL WITH NULLS.
  - PARAMETER STYLE DB2DARI is supported.

  The following syntax is accepted as the default behavior:
  - ASUTIME NO LIMIT
  - NO COLLID
  - STAY RESIDENT NO
  - CCSID UNICODE in a Unicode database
  - CCSID ASCII in a non-Unicode database if PARAMETER CCSID UNICODE is not specified

Examples

- *Example 1:* Create the procedure definition for a procedure, written in Java, that is passed a part number and that returns the cost of the part and the quantity that is currently available.

```
CREATE PROCEDURE PARTS_ON_HAND (IN PARTNUM INTEGER,
    OUT COST DECIMAL(7,2),
    OUT QUANTITY INTEGER)
EXTERNAL NAME 'parts.onhand'
LANGUAGE JAVA PARAMETER STYLE JAVA
```

- *Example 2:* Create the procedure definition for a procedure, written in C, that is passed an assembly number and returns the number of parts that make up the assembly, total part cost, and a result set that lists the part numbers, quantity, and unit cost of each part.

```
CREATE PROCEDURE ASSEMBLY_PARTS (IN ASSEMBLY_NUM INTEGER,
    OUT NUM_PARTS INTEGER,
    OUT COST DOUBLE)
  EXTERNAL NAME 'parts!assembly'
  DYNAMIC RESULT SETS 1 NOT FENCED
  LANGUAGE C PARAMETER STYLE GENERAL
```

CREATE PROCEDURE (sourced)

The CREATE PROCEDURE (sourced) statement defines a procedure (the *sourced procedure*) that is based on another procedure (the *source procedure*). In a federated system, a *federated procedure* is a sourced procedure whose source procedure is at a supported data source.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

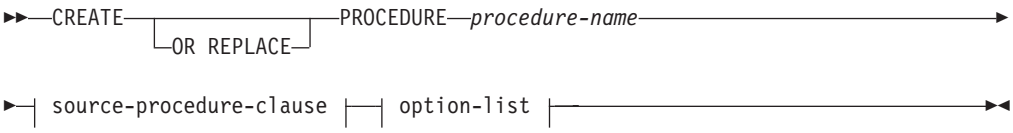
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the schema name of the procedure does not refer to an existing schema
- CREATEIN privilege on the schema, if the schema name of the procedure refers to an existing schema
- DBADM authority

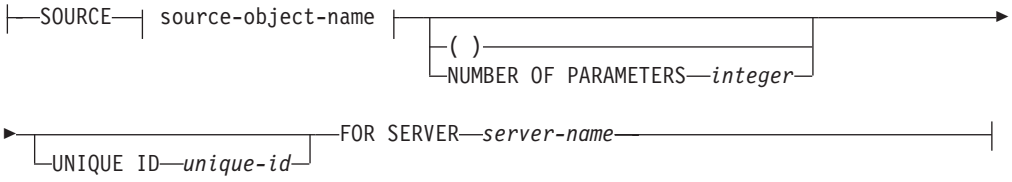
For data sources that require a user mapping, the privileges held at the data source by the authorization ID of the statement must include the privilege to select the procedure's description from the remote catalog tables.

To replace an existing procedure, the authorization ID of the statement must be the owner of the existing procedure (SQLSTATE 42501).

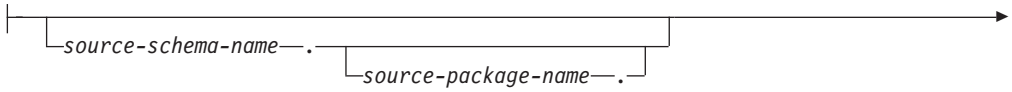
Syntax

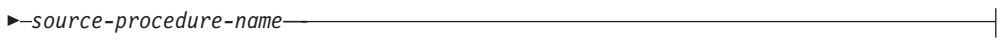


source-procedure-clause:

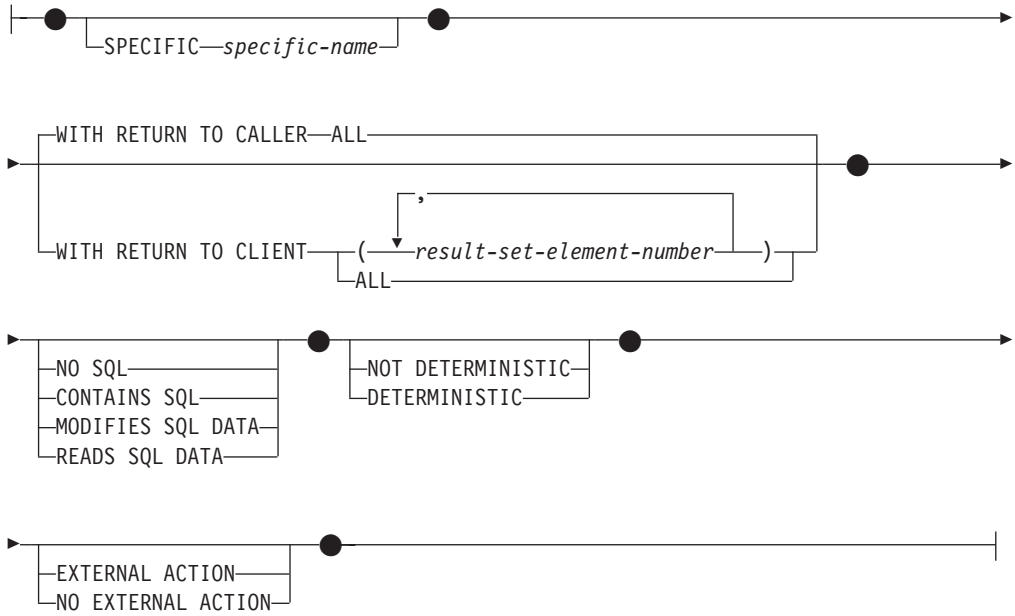


source-object-name:





option-list:



Description

OR REPLACE

Specifies to replace the definition for the procedure if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the procedure are not affected. This option can be specified only by the owner of the object. This option is ignored if a definition for the procedure does not exist at the current server. To replace an existing procedure, the specific name and procedure name of the new definition must be the same as the specific name and procedure name of the old definition, or the signature of the new definition must match the signature of the old definition. Otherwise, a new procedure is created.

procedure-name

Names the sourced procedure being defined. It is a qualified or unqualified name that designates a procedure. The unqualified form of *procedure-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters, must not identify a procedure that is described in the catalog (SQLSTATE 42723). The unqualified name, together with the number of parameters, need not be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

In a federated system, *procedure-name* is the name of the procedure on the federated server.

SOURCE *source-object-name*
Specifies the source procedure that is used by the procedure being defined. In a federated system, the source procedure is a procedure that is located at a supported data source.

source-schema-name
Identifies the schema name of the source procedure. If a schema name is used to identify the source procedure, the *source-schema-name* must be specified in the CREATE PROCEDURE (Sourced) statement. If the *source-schema-name* contains any special or lowercase characters, it must be enclosed by double quotation marks.

source-package-name
Identifies the package name of the source procedure. The *source-package-name* applies only to Oracle data sources. If a package name is used to identify the source procedure, the *source-package-name* must be specified in the CREATE PROCEDURE (Sourced) statement. If the *source-package-name* contains any special or lowercase characters, it must be enclosed by double quotation marks.

source-procedure-name
Identifies the procedure name of the source procedure. If the *source-procedure-name* contains any special or lowercase characters, it must be enclosed by double quotation marks.

( )
Indicates that the number of parameters is zero.

NUMBER OF PARAMETERS *integer*
Specifies the number of parameters for the source procedure. The minimum value for *integer* is 0, and the maximum value is 32 767.

UNIQUE ID *string-constant*
Provides a way to uniquely identify the source procedure when there are multiple procedures at the data source with the identical name, schema, and number of parameters. The *string-constant* value, which has a maximum length of 128, is interpreted uniquely by each data source.

FOR SERVER *server-name*
Specifies a server definition that was registered using the CREATE SERVER statement.

SPECIFIC *specific-name*
Provides a unique name for the instance of the sourced procedure that is being defined. This specific name can be used when altering, dropping, or commenting on the sourced procedure. This name can never be used to invoke the sourced procedure. The unqualified form of *specific-name* is an SQL identifier. The qualified form of *specific-name* is a *schema-name* followed by a period and an SQL identifier. The *specific-name* value, including the implicit or explicit qualifier, must not identify another procedure instance that exists at the application server; otherwise an error is returned (SQLSTATE 42710).

The *specific-name* can be the same as an existing *procedure-name*.

If no qualifier is specified, the qualifier that was used for *procedure-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier for *procedure-name*, or an error is returned (SQLSTATE 42882).

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is 'SQL' followed by a character timestamp: 'SQL*yymmddhhmmssxxx*'.

WITH RETURN TO CALLER or WITH RETURN TO CLIENT
Indicates where the result sets from the source procedure are handled. If the source procedure is not from an Oracle data source, the only one result set is returned to the caller or client; and if the source procedure is coded to return more than one result set, only the first result set is returned to the caller or client. The default is WITH RETURN TO CALLER.

WITH RETURN TO CALLER ALL
Specifies that all result sets from the source procedure are returned to the caller.

WITH RETURN TO CLIENT
Indicates which result sets from the source procedure are returned directly to the client application. The dynamic result set value at the data source must be greater than 0 for a result set to be returned.

(result-set-element-number, ...)
Specifies a non-empty list of result sets to return to the client application (SQLSTATE 42601). A *result-set-element-number* identifies a result set based on the order the result sets are returned, where 1 identifies the first result set, 2 the second result set, and so on. A *result-set-element-number* greater than the total number of result sets returned is ignored. Each *result-set-element-number* must be an integer value greater than zero (SQLSTATE 42815), and must not exceed the value of a small integer constant (SQLSTATE 42820). The list of result sets to return to the client application must not contain duplicate values and must be specified in ascending order (SQLSTATE 42815). Result sets are always processed in the order they are returned from the source procedure.

Result sets that are not identified in the list to return to client application are returned to the caller.

Note: This list of result sets to return to the client application must only be used with source procedures that are known to consistently return result sets that are intended for the client in the same position in the list of result sets each time they are executed. It is possible for a source procedure to return different sets of result sets each time it is executed, depending on the internal logic of the procedure. If this is the case, then specify either WITH RETURN TO CALLER ALL or WITH RETURN TO CLIENT ALL instead, and code the application to handle this case.

ALL
Specifies all result sets from the source procedure are returned to the client.

NO SQL, CONTAINS SQL, MODIFIES SQL DATA, READS SQL DATA
Indicates the level of data access for SQL statements that are included in the sourced procedure. Because the source procedure for the sourced procedure is not located on the federated server, the specified level is not enforced during execution of the source procedure at the data source. If there is discrepancy between what is specified for the sourced procedure and what the source procedure actually does at the data source, data inconsistency might occur. If this option is not explicitly specified, the value for the source procedure is

used. If this option is not available at the data source, the default is MODIFIES SQL DATA. If this option is explicitly specified but does not match the value for the source procedure, an error is returned (SQLSTATE 428GS).

DETERMINISTIC or NOT DETERMINISTIC
Specifies whether the sourced procedure always returns the same results for given argument values (DETERMINISTIC), or whether the sourced procedure depends on some stated values that affect the results (NOT DETERMINISTIC). A DETERMINISTIC sourced procedure must always return the same result from successive invocations with identical inputs. This clause currently does not impact the processing of the procedure. If this option is not explicitly specified, the value for the source procedure is used. If this option is not available at the data source, the default is NOT DETERMINISTIC. If this option is explicitly specified, but does not match the value for the source procedure, an error is returned (SQLSTATE 428GS).

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the sourced procedure takes some action that changes the state of an object that is not managed by the database manager (EXTERNAL ACTION), or does not (NO EXTERNAL ACTION). If the NO EXTERNAL ACTION clause is specified, the federated database uses optimization that assumes that the sourced procedure has no external impact. If this option is not explicitly specified, the value for the source procedure is used. If this option is not available at the data source, the default is EXTERNAL ACTION. If this option is explicitly specified but does not match the value for the source procedure, an error is returned (SQLSTATE 428GS).

Rules

- If the *source-object-name*, along with the NUMBER OF PARAMETERS and UNIQUE ID clauses do not identify a procedure at the data source, an error is returned (SQLSTATE 42883); if more than one procedure is identified, an error is returned (SQLSTATE 42725).
- If the UNIQUE ID clause is specified and the data source does not support unique IDs, an error is returned (SQLSTATE 42883).

Notes

- Before a federated procedure can be registered for a data source, the federated server must be configured to access that data source. This configuration includes: registering the wrapper for the data source, creating the server definition for the data source, and creating the user mappings between the federated server and the data source server for the data sources that require user mapping.
- ***Creating procedures that are initially invalid:*** If an object referenced in the procedure body does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the procedure will still be created successfully. The procedure will be marked invalid and will be revalidated the next time it is invoked.
- Unlike SQL and external procedures defined at the federated server, federated procedures do not inherit the special registers of the caller, even those whose *remote-object-name* refers to a procedure on a DB2 data source.
- If the definition of the source procedure is changed (for example, a parameter data type is changed), the federated procedure should be dropped and recreated; otherwise, errors might occur when the federated procedure is invoked.

- If the length of the source procedure parameter is longer than 128, the parameter name of the federated procedure is truncated to 128 bytes.
- ***Compatibilities***: The DataJoiner syntax for Create Stored Procedure Nickname is not supported. In the new Version 9 syntax, parameter type mapping is handled similarly to nicknames: A catalog look-up determines the remote data type. The local parameter type is determined through forward type mapping.

Examples

- *Example 1:* Create a federated procedure named FEDEMPLOYEE for an Oracle procedure named EMPLOYEE, using the remote schema name USER1, the remote package name P1 at the federated server S1, and returning the result set to the client.

```
CREATE PROCEDURE FEDEMPLOYEE SOURCE USER1.P1.EMPLOYEE
  FOR SERVER S1 WITH RETURN TO CLIENT ALL
```

- *Example 2:* Create a federated procedure named FEDSALARYSTAT for an Oracle procedure named SALARYSTAT, using the remote schema name USER1, the remote package name P1 at the federated server S1, and returning the first and the third result set to the client, and remaining result sets to the caller.

```
CREATE OR REPLACE PROCEDURE FEDSALARYSTAT SOURCE USER1.P1.SALARYSTAT
FOR SERVER S1 WITH RETURN TO CLIENT(1,3)
```

CREATE PROCEDURE (SQL)

The CREATE PROCEDURE (SQL) statement defines an SQL procedure at the current server.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

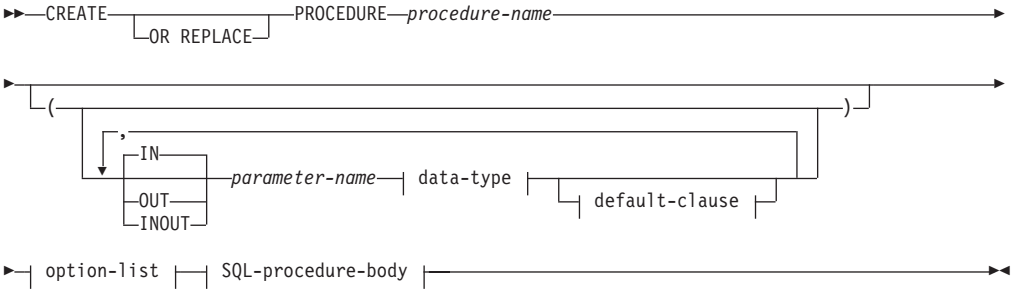
- If the implicit or explicit schema name of the procedure does not exist, IMPLICIT_SCHEMA authority on the database.
- If the schema name of the procedure refers to an existing schema, CREATEIN privilege on the schema.
- DBADM authority

The privileges held by the authorization ID of the statement must also include all of the privileges necessary to invoke the SQL statements that are specified in the procedure body.

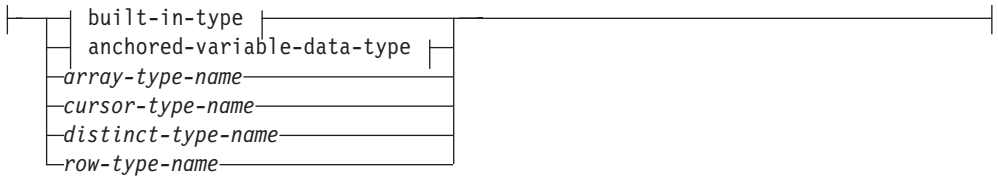
To replace an existing procedure, the authorization ID of the statement must be the owner of the existing procedure (SQLSTATE 42501).

Group privileges are not considered for any table or view specified in the CREATE PROCEDURE (SQL) statement.

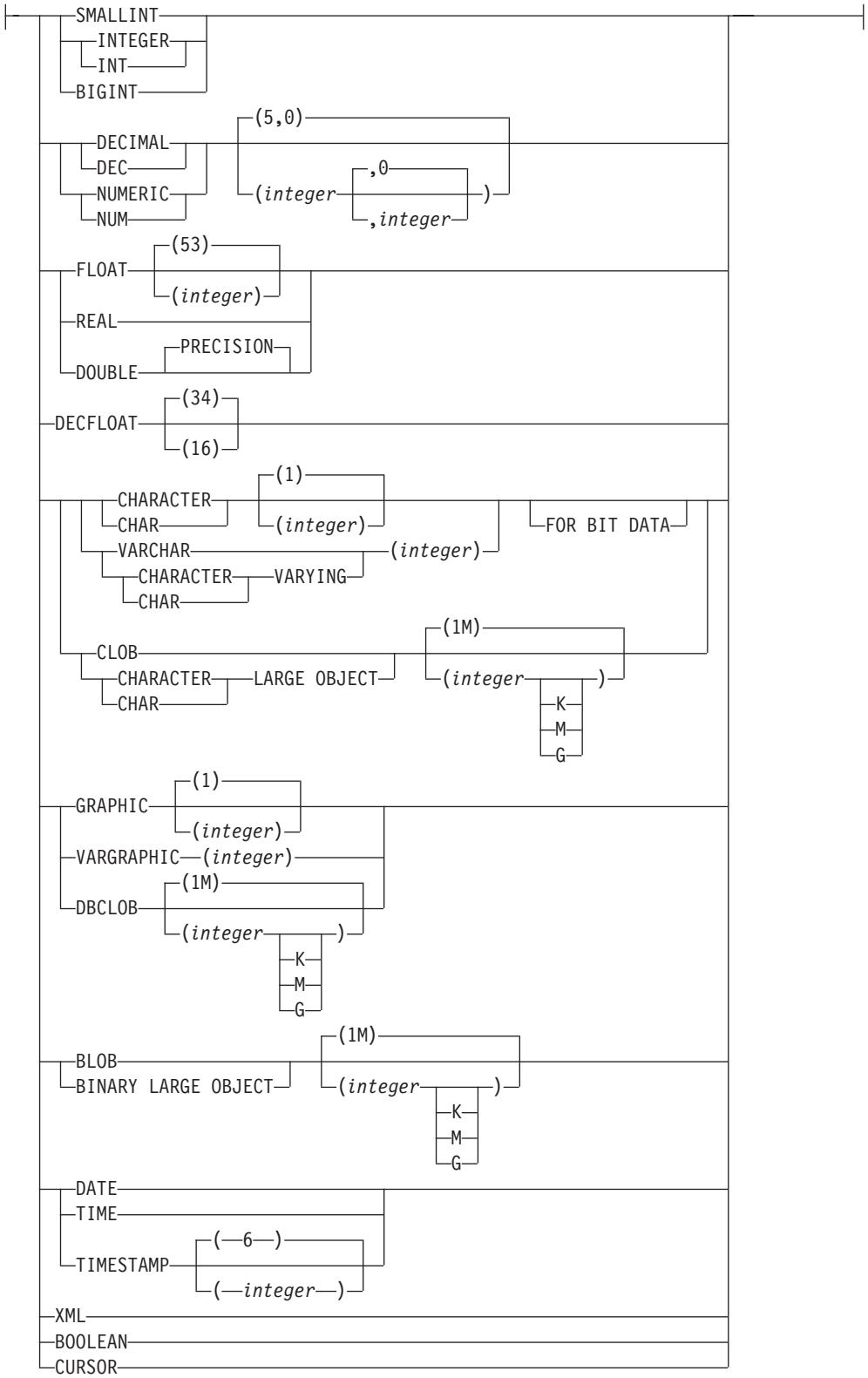
Syntax



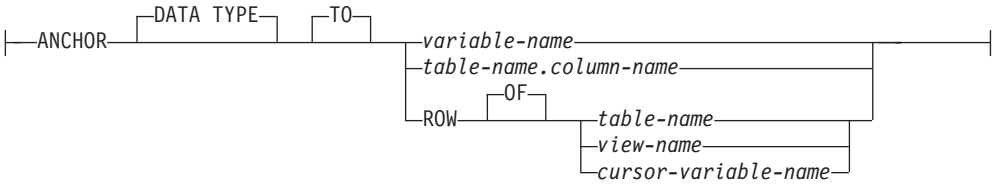
data-type:



built-in-type:

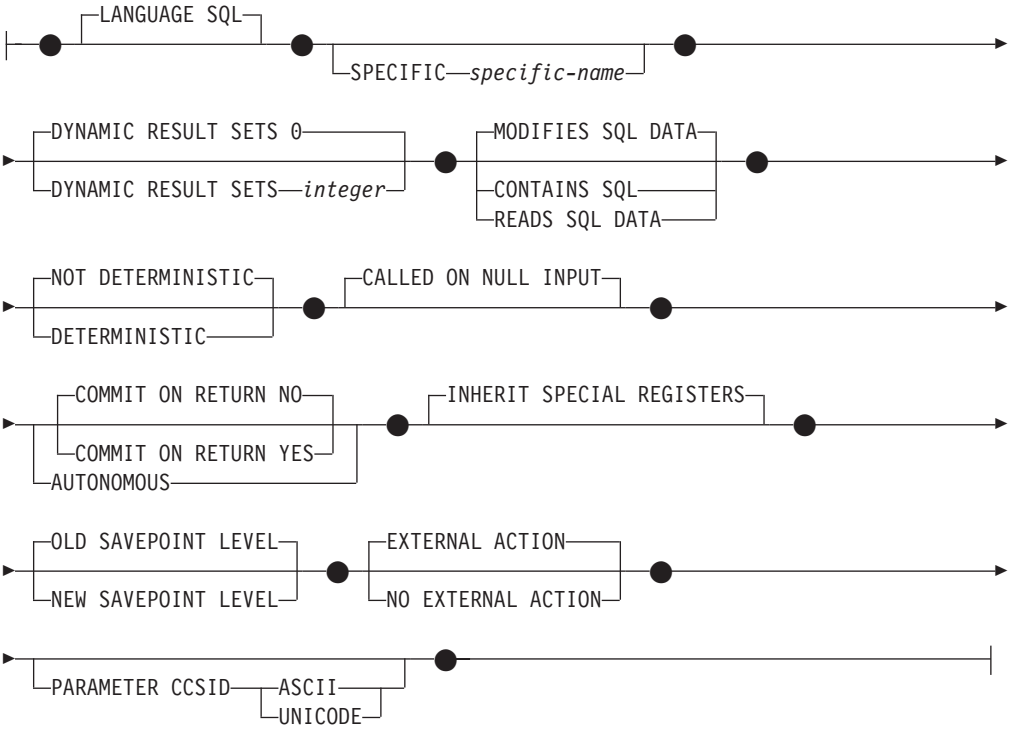


anchored-data-type:



default-clause:

option-list:



SQL-procedure-body:

Description

OR REPLACE

Specifies to replace the definition for the procedure if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted

on the procedure are not affected. This option can be specified only by the owner of the object. This option is ignored if a definition for the procedure does not exist at the current server. To replace an existing procedure, the specific name and procedure name of the new definition must be the same as the specific name and procedure name of the old definition, or the signature of the new definition must match the signature of the old definition. Otherwise, a new procedure is created.

procedure-name
Names the procedure being defined. It is a qualified or unqualified name that designates a procedure. The unqualified form of *procedure-name* is an SQL identifier. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters, must not identify a procedure described in the catalog (SQLSTATE 42723). The unqualified name, together with the number of parameters, is unique within its schema, but does not need to be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS'; otherwise, an error is returned (SQLSTATE 42939).

*(***IN** | **OUT** | **INOUT** *parameter-name data-type default-clause,...)*
Identifies the parameters of the procedure, and specifies the mode, name, data type, and optional default value of each parameter. One entry in the list must be specified for each parameter that the procedure will expect.

It is possible to register a procedure that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example:

```
CREATE PROCEDURE SUBWOOFER() ...
```

No two identically-named procedures within a schema are permitted to have exactly the same number of parameters. A duplicate signature raises an SQL error (SQLSTATE 42723).

For example, given the statements:

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART_NAME CHAR(35)) ...
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

the second statement will fail because the number of parameters in the procedure is the same, even if the data types are not.

IN | **OUT** | **INOUT**
Specifies the mode of the parameter.

If an error is returned by the procedure, OUT parameters are undefined and INOUT parameters are unchanged.

IN Identifies the parameter as an input parameter to the procedure. Any changes made to the parameter within the procedure are not available to the calling SQL application when control is returned. The default is IN.

OUT Identifies the parameter as an output parameter for the procedure.

INOUT
Identifies the parameter as both an input and output parameter for the procedure.

parameter-name
Specifies the name of the parameter. The parameter name must be unique for the procedure (SQLSTATE 42734).

data-type
Specifies the data type of the parameter. A structured type or reference type cannot be specified (SQLSTATE 429BB).

built-in-type
Specifies a built-in data type. For a more complete description of each built-in data type except BOOLEAN and CURSOR, which cannot be specified for a table, see "CREATE TABLE".

BOOLEAN
For a Boolean.

CURSOR
For a reference to an underlying cursor.

anchored-data-type
Identifies another object used to define the data type. The data type of the anchor object has the same limitations that apply to specifying the data type directly, or in the case of a row, to creating a row type.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies a global variable. The data type of the global variable is used as the data type for *parameter-name*.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for *parameter-name*.

ROW OF *table-name* **or** *view-name*
Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data type of *parameter-name* is an unnamed row type.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following elements (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a *select-statement* where all the result columns are named.

If the cursor type of the cursor variable is not strongly-typed using a named row type, the data type of *parameter-name* is an unnamed row type.

array-type-name
Specifies the name of a user-defined array type. If *array-type-name* is specified without a schema name, the array type is resolved by searching the schemas in the SQL path.

cursor-type-name
Specifies the name of a cursor type. If *cursor-type-name* is specified without a schema name, the cursor type is resolved by searching the schemas in the SQL path.

distinct-type-name
Specifies the name of a distinct type. The length, precision, and scale of the parameter are, respectively, the length, precision, and scale of the source type of the distinct type. A distinct type parameter is passed as the source type of the distinct type. If *distinct-type-name* is specified without a schema name, the distinct type is resolved by searching the schemas in the SQL path.

row-type-name
Specifies the name of a user-defined row type. The fields of the parameter are the fields of the row type. If *row-type-name* is specified without a schema name, the row type is resolved by searching the schemas in the SQL path.

DEFAULT
Specifies a default value for the parameter. The default can be a constant, a special register, a global variable, an expression or the keyword NULL. The special registers that can be specified as the default are that same as those that can be specified for a column default (see *default-clause* in the CREATE TABLE statement). Other special registers can be specified as the default by using an expression.

The *expression* can be any expression of the type described in "Expressions". If a default value is not specified, the parameter has no default and the corresponding argument cannot be omitted on invocation of the procedure. The maximum size of the *expression* is 64K bytes.

The default expression must not modify SQL data (SQLSTATE 428FL or SQLSTATE 429BL). The expression must be assignment compatible to the parameter data type (SQLSTATE 42821).

A default cannot be specified in the following situations:

- For INOUT or OUT parameters (SQLSTATE 42601)
- For a parameter of type ARRAY, ROW, or CURSOR (SQLSTATE 429BB)

SPECIFIC *specific-name*
Provides a unique name for the instance of the procedure that is being defined. This specific name can be used when altering, dropping, or commenting on the procedure. It can never be used to invoke the procedure. The unqualified form of *specific-name* is an SQL identifier. The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another procedure instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* can be the same as an existing *procedure-name*.

If no qualifier is specified, the qualifier that was used for *procedure-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier for *procedure-name*, or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is 'SQL' followed by a character timestamp: 'SQL*yymmddhhmmssxxx*'.

DYNAMIC RESULT SETS *integer*
Indicates the estimated upper bound of returned result sets for the procedure.

CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA
Indicates the level of data access for SQL statements included in the procedure.

CONTAINS SQL
Indicates that SQL statements that neither read nor modify SQL data can be executed by the procedure (SQLSTATE 38004 or 42985). Statements that are not supported in procedures might return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA
Indicates that some SQL statements that do not modify SQL data can be included in the procedure (SQLSTATE 38002 or 42985). Statements that are not supported in procedures might return a different error (SQLSTATE 38003 or 42985).

MODIFIES SQL DATA
Indicates that the procedure can execute any SQL statement except statements that are not supported in procedures (SQLSTATE 38003 or 42985).

If the BEGIN ATOMIC clause is used in a compound SQL procedure, the procedure can only be created if it is defined as MODIFIES SQL DATA.

DETERMINISTIC or NOT DETERMINISTIC
This clause specifies whether the procedure always returns the same results for given argument values (DETERMINISTIC) or whether the procedure depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC procedure must always return the same result from successive invocations with identical inputs.

This clause currently does not impact processing of the procedure.

CALLED ON NULL INPUT
CALLED ON NULL INPUT always applies to procedures. This means that the procedure is called regardless of whether any arguments are null. Any OUT or INOUT parameter can return a null value or a normal (non-null) value. Responsibility for testing for null argument values lies with the procedure.

COMMIT ON RETURN
Indicates whether a commit is to be issued on return from the procedure. The default is NO.

NO A commit is not issued when the procedure returns.

YES
A commit is issued when the procedure returns if a positive SQLCODE is returned by the CALL statement

The commit operation includes the work that is performed by the calling application process and the procedure.

If the procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

AUTONOMOUS
Indicates the procedure should execute in its own autonomous transaction scope.

INHERIT SPECIAL REGISTERS
This optional clause specifies that updatable special registers in the procedure will inherit their initial values from the environment of the invoking statement.

For a routine invoked in a nested object (for example a trigger or view), the initial values are inherited from the runtime environment (not inherited from the object definition).

No changes to the special registers are passed back to the caller of the procedure.

Non-updatable special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore set to their default values.

OLD SAVEPOINT LEVEL or NEW SAVEPOINT LEVEL
Specifies whether or not this procedure establishes a new savepoint level for savepoint names and effects. OLD SAVEPOINT LEVEL is the default behavior. For more information about savepoint levels, see "Rules" in "SAVEPOINT".

LANGUAGE SQL
This clause is used to specify that the procedure body is written in the SQL language.

EXTERNAL ACTION or NO EXTERNAL ACTION
Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not (NO EXTERNAL ACTION). The default is EXTERNAL ACTION. If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the procedure. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE
Specifies that character data is in UTF-8, and that graphic data is in UCS-2. If the database is not a Unicode database, PARAMETER CCSID UNICODE cannot be specified (SQLSTATE 56031).

SQL-procedure-body
Specifies the SQL statement that is the body of the SQL procedure.

See *SQL-procedure-statement* in "Compound SQL (Compiled)" statement.

Rules

- ***Autonomous routine restrictions:*** Autonomous routines cannot return result sets and do not support the following data types (SQLSTATE 428H2):
  - User-defined cursor types
  - User-defined structured types
  - XML as IN, OUT, and INOUT parameters

  Session variables of cursor types cannot be referenced within the autonomous scope.
- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row

definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

- ***Use of cursor and row types:*** A procedure that uses a cursor type or row type for a parameter can only be invoked from within a compound SQL (compiled) statement (SQLSTATE 428H2), except for JDBC which can invoke a procedure with OUT parameters that have a cursor type.

Notes

- Creating a procedure with a schema name that does not already exist will result in the implicit creation of that schema, provided that the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- A procedure that is called from within a compound SQL (inlined) statement will execute as if it were created specifying NEW SAVEPOINT LEVEL, even if OLD SAVEPOINT LEVEL was specified or defaulted to when the procedure was created.
- ***Creating procedures that are initially invalid:*** If an object referenced in the procedure body does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the procedure will still be created successfully. The procedure will be marked invalid and will be revalidated the next time it is invoked.
- ***Setting of the default value:*** Parameters of a procedure that are defined with a default value are set to their default value when the procedure is invoked, but only if a value is not supplied for the corresponding argument, or is specified as DEFAULT, when the procedure is invoked.
- ***Privileges:*** The definer of a procedure always receives the EXECUTE privilege WITH GRANT OPTION on the procedure, as well as the right to drop the procedure.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - RESULT SETS can be specified in place of DYNAMIC RESULT SETS.
  - NULL CALL can be specified in place of CALLED ON NULL INPUT.

  The following syntax is accepted as the default behavior:
  - ASUTIME NO LIMIT
  - NO COLLID
  - STAY RESIDENT NO

Example

Create an SQL procedure that returns the median staff salary. Return a result set containing the name, position, and salary of all employees who earn more than the median salary.

```
CREATE PROCEDURE MEDIAN_RESULT_SET (OUT medianSalary DOUBLE)
  RESULT SETS 1
  LANGUAGE SQL
BEGIN
  DECLARE v_numRecords INT DEFAULT 1;
  DECLARE v_counter INT DEFAULT 0;

  DECLARE c1 CURSOR FOR
    SELECT CAST(salary AS DOUBLE)
      FROM staff
```

```
    ORDER BY salary;
  DECLARE c2 CURSOR WITH RETURN FOR
    SELECT name, job, CAST(salary AS INTEGER)
      FROM staff
      WHERE salary > medianSalary
      ORDER BY salary;

  DECLARE EXIT HANDLER FOR NOT FOUND
    SET medianSalary = 6666;

  SET medianSalary = 0;
  SELECT COUNT(*) INTO v_numRecords
    FROM STAFF;
  OPEN c1;
  WHILE v_counter < (v_numRecords / 2 + 1)
  DO
    FETCH c1 INTO medianSalary;
    SET v_counter = v_counter + 1;
  END WHILE;
  CLOSE c1;
  OPEN c2;
END
```

CREATE ROLE

The CREATE ROLE statement defines a role at the current server.

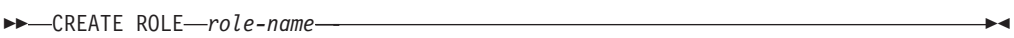
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

role-name
Names the role. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must not identify an existing role at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' and must not be 'ACCESSCTRL', 'DATAACCESS', 'DBADM', 'NONE', 'NULL', 'PUBLIC', 'SECADM', 'SQLADM', or 'WLMADM' (SQLSTATE 42939).

Example

Create a role named DOCTOR.

```
CREATE ROLE DOCTOR
```

CREATE SCHEMA

The CREATE SCHEMA statement defines a schema. It is also possible to create some objects and grant privileges on objects within the statement.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

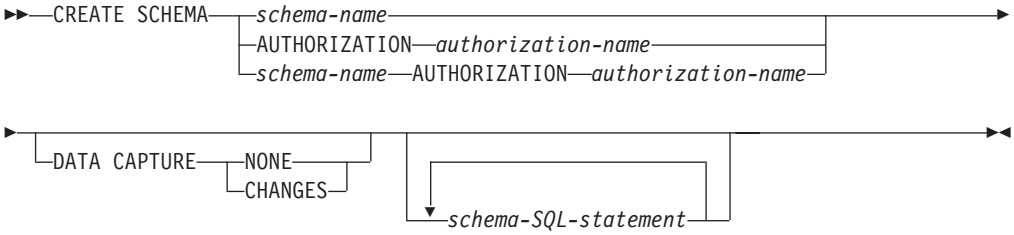
An authorization ID that holds DBADM authority can create a schema with any valid *schema-name* or *authorization-name*.

An authorization ID that does not hold DBADM authority can only create a schema with a *schema-name* or *authorization-name* that matches the authorization ID of the statement.

If the statement includes a *schema-SQL-statement*, the privileges held by the *authorization-name* (which, if not specified, defaults to the authorization ID of the statement) must include at least one of the following authorities:

- The privileges required to perform each *schema-SQL-statement*
- DBADM authority

Syntax



Description

schema-name
Names the schema. The name must not identify a schema already described in the catalog (SQLSTATE 42710). The name cannot begin with 'SYS' (SQLSTATE 42939). The owner of the schema is the authorization ID that issued the statement.

AUTHORIZATION *authorization-name*
Identifies the user who is the owner of the schema. The value of *authorization-name* is also used to name the schema. The *authorization-name* must not identify a schema already described in the catalog (SQLSTATE 42710).

schema-name **AUTHORIZATION** *authorization-name*
Identifies a schema called *schema-name*, whose owner is *authorization-name*. The *schema-name* must not identify a schema already described in the catalog (SQLSTATE 42710). The *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

DATA CAPTURE
Indicates whether extra information for data replication is to be written to the log. The default is determined based on the value of database configuration parameter **dft_schemas_dcc**. If the value is "Yes" the default is CHANGES, otherwise the default is NONE.

NONE
Indicates that no extra information for data replication will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this schema will be written to the log. This option is required if this schema will be replicated and a replication capture program is used to capture changes for this schema from the log.

schema-SQL-statement
SQL statements that can be included as part of the CREATE SCHEMA statement are:

- CREATE TABLE statement, excluding typed tables and materialized query tables
- CREATE VIEW statement, excluding typed views
- CREATE INDEX statement
- COMMENT statement
- GRANT statement

Notes

- The owner of the schema is determined as follows:
  - If an AUTHORIZATION clause is specified, the specified *authorization-name* is the schema owner
  - If an AUTHORIZATION clause is not specified, the authorization ID that issued the CREATE SCHEMA statement is the schema owner.
- The schema owner is assumed to be a user (not a group).
- When the schema is explicitly created with the CREATE SCHEMA statement, the schema owner is granted CREATEIN, DROPIN, and ALTERIN privileges on the schema with the ability to grant these privileges to other users.
- The definer of any object created as part of the CREATE SCHEMA statement is the schema owner. The schema owner is also the grantor for any privileges granted as part of the CREATE SCHEMA statement.
- Unqualified object names in any SQL statement within the CREATE SCHEMA statement are implicitly qualified by the name of the created schema.
- If the CREATE statement contains a qualified name for the object being created, the schema name specified in the qualified name must be the same as the name of the schema being created (SQLSTATE 42875). Any other objects referenced within the statements may be qualified with any valid schema name.
- It is recommended not to use "SESSION" as a schema name. Since declared temporary tables must be qualified by "SESSION", it is possible to have an application declare a temporary table with a name identical to that of a persistent table. An SQL statement that references a table with the schema name "SESSION" will resolve (at statement compile time) to the declared temporary table rather than a persistent table with the same name. Since an SQL statement is compiled at different times for static embedded and dynamic embedded SQL statements, the results depend on when the declared temporary table is defined. If persistent tables, views or aliases are not defined with a schema name of "SESSION", these issues do not require consideration.

- Setting the DATA CAPTURE attribute at the schema level causes newly created tables to inherit the DATA CAPTURE attribute from the schema if one is not specified at the table level.

Examples

- *Example 1:* As a user with DBADM authority, create a schema called RICK with the user RICK as the owner.

```
CREATE SCHEMA RICK AUTHORIZATION RICK
```

- *Example 2:* Create a schema that has an inventory part table and an index over the part number. Give authority on the table to user JONES.

```
CREATE SCHEMA INVENTRY

  CREATE TABLE PART (PARTNO   SMALLINT NOT NULL,
                     DESCR    VARCHAR(24),
                     QUANTITY INTEGER)

  CREATE INDEX PARTIND ON PART (PARTNO)

  GRANT ALL ON PART TO JONES
```

- *Example 3:* Create a schema called PERS with two tables that each have a foreign key that references the other table. This is an example of a feature of the CREATE SCHEMA statement that allows such a pair of tables to be created without the use of the ALTER TABLE statement.

```
CREATE SCHEMA PERS

  CREATE TABLE ORG (DEPTNUMB  SMALLINT NOT NULL,
                    DEPTNAME VARCHAR(14),
                    MANAGER  SMALLINT,
                    DIVISION VARCHAR(10),
                    LOCATION VARCHAR(13),
                    CONSTRAINT PKEYDNO
                      PRIMARY KEY (DEPTNUMB),
                    CONSTRAINT FKEYMGR
                      FOREIGN KEY (MANAGER)
                      REFERENCES STAFF (ID) )

  CREATE TABLE STAFF (ID        SMALLINT NOT NULL,
                      NAME     VARCHAR(9),
                      DEPT     SMALLINT,
                      JOB      VARCHAR(5),
                      YEARS    SMALLINT,
                      SALARY   DECIMAL(7,2),
                      COMM     DECIMAL(7,2),
                      CONSTRAINT PKEYID
                        PRIMARY KEY (ID),
                      CONSTRAINT FKEYDNO
                        FOREIGN KEY (DEPT)
                        REFERENCES ORG (DEPTNUMB) )
```

CREATE SECURITY LABEL COMPONENT

The CREATE SECURITY LABEL COMPONENT statement defines a component that is to be used as part of a security policy.

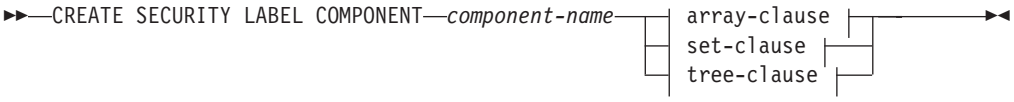
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

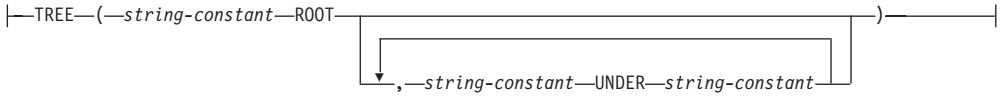
Syntax



array-clause:

set-clause:

tree-clause:



Description

component-name
Names the security label component. This is a one-part name. The name must not identify an existing security label component at the current server (SQLSTATE 42710).

ARRAY
Specifies an ordered set of elements.

string-constant,...
One or more string constant values that make up the set of valid values for this security label component. The order in which the array elements

appear is important. The first element ranks higher than the second element. The second element ranks higher than the third element and so on.

SET
Specifies an unordered set of elements.

string-constant,...
One or more string constant values that make up the set of valid values for this security label component. The order of the elements is not important.

TREE
Specifies a tree structure of node elements.

string-constant
One or more string constant values that make up the set of valid values for this security label component.

ROOT
Specifies that the *string-constant* that follows the keyword is the root node element of the tree.

UNDER
Specifies that the *string-constant* before the **UNDER** keyword is a child of the *string-constant* that follows the **UNDER** keyword. An element must be defined as either being the root element or as being the child of another element before it can be used as a parent, otherwise an error (SQLSTATE 42704) is returned.

Rules

These rules apply to all three types of component (ARRAY, SET, and TREE):

- Element names cannot contain any of these characters:
  - Opening parenthesis - (
  - Closing parenthesis -)
  - Comma - ,
  - Colon - :
- An element name can have no more than 32 bytes (SQLSTATE 42622).
- If a security label component is a set or a tree, no more than 64 elements can be part of that component.
- A CREATE SECURITY LABEL COMPONENT statement can specify at most 65 535 elements for a security label component of type array.
- No element name can be used more than once in the same component (SQLSTATE 42713).

Examples

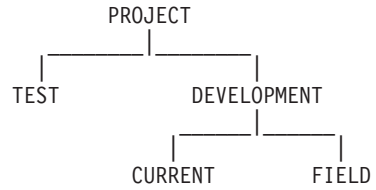
- *Example 1:* Create an ARRAY type security label component named LEVEL. The component has the following four elements, listed in order of decreasing rank: Top Secret, Secret, Classified, and Unclassified.

```
CREATE SECURITY LABEL COMPONENT LEVEL
  ARRAY ['Top Secret', 'Secret', 'Classified', 'Unclassified']
```

- *Example 2:* Create a SET type security label component named COMPARTMENTS. The component has the following three elements: Research, Analysis, and Collection.

```
CREATE SECURITY LABEL COMPONENT COMPARTMENTS
  SET {'Collection', 'Research', 'Analysis'}
```

- *Example 3:* Create a TREE type security label component named GROUPS. GROUPS has five elements: PROJECT, TEST, DEVELOPMENT, CURRENT, AND FIELD. The following diagram shows the relationship of these elements to one another:




```
CREATE SECURITY LABEL COMPONENT GROUPS
  TREE (
    'PROJECT' ROOT,
    'TEST' UNDER 'PROJECT',
    'DEVELOPMENT' UNDER 'PROJECT',
    'CURRENT' UNDER 'DEVELOPMENT',
    'FIELD' UNDER 'DEVELOPMENT'
  )
```

CREATE SECURITY LABEL

The CREATE SECURITY LABEL statement defines a security label.

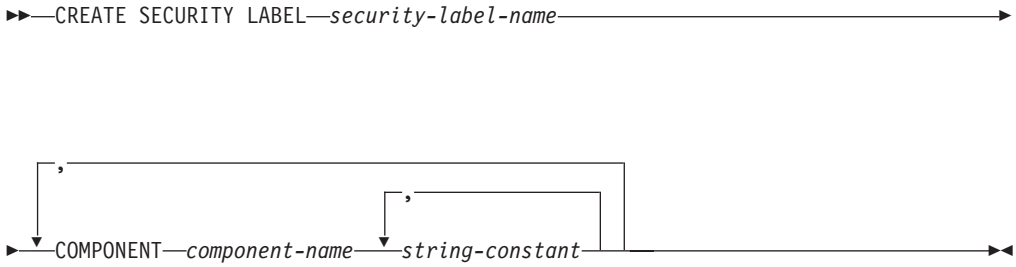
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

security-label-name
Names the security label. The name must be qualified with a security policy (SQLSTATE 42704), and must not identify an existing security label for this security policy (SQLSTATE 42710).

COMPONENT *component-name*
Specifies the name of a security label component. If the component is not part of the security policy *security-policy-name*, an error is returned (SQLSTATE 4274G). If a component is specified twice in the same statement, an error is returned (SQLSTATE 42713).

string-constant,...
Specifies a valid element for the security component. A valid element is one that was specified when the security component was created. If the element is invalid, an error is returned (SQLSTATE 4274F).

Examples

- *Example 1:* Create a security label named EMPLOYEESECLABEL that is part of the DATA_ACCESS security policy, and that has the element Top Secret for the LEVEL component and the elements Research and Analysis for the COMPARTMENTS component.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
  COMPONENT LEVEL 'Top Secret',
  COMPONENT COMPARTMENTS 'Research', 'Analysis'
```

- *Example 2:* Create a security label named EMPLOYEESECLABELREAD that has the element Top Secret for the LEVEL component and the element Research for the COMPARTMENTS component.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELREAD
  COMPONENT LEVEL 'Top Secret',
  COMPONENT COMPARTMENTS 'Research'
```

- *Example 3:* Create a security label named EMPLOYEESECLABELWRITE that has the element Analysis for the COMPARTMENTS component and a null value for the LEVEL component. Assume that the security policy named DATA_ACCESS is the same security policy that is used in examples 1 and 2.

  ```
  CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELWRITE
    COMPONENT COMPARTMENTS 'Analysis'
  ```

- *Example 4:* Create a security label named BEGINNER that is part of an existing CLASSPOLICY security policy, and that has the element Trainee for the TRUST component and the element Morning for the SECTIONS component.

  ```
  CREATE SECURITY LABEL CLASSPOLICY.BEGINNER
    COMPONENT TRUST 'Trainee',
    COMPONENT SECTIONS 'Morning'
  ```

CREATE SECURITY POLICY

The CREATE SECURITY POLICY statement defines a security policy.

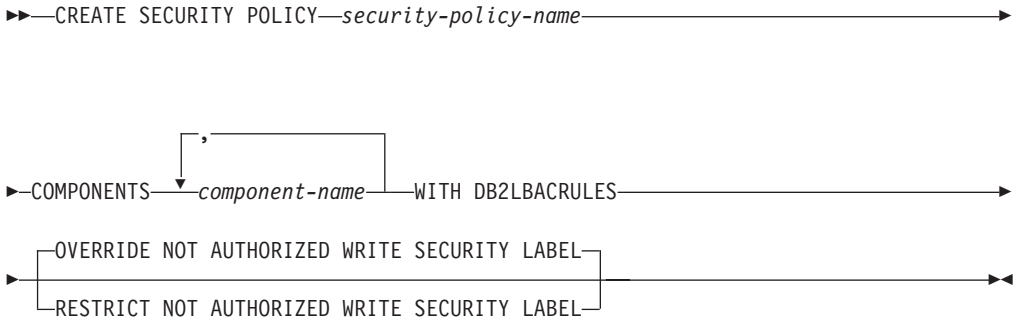
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

security-policy-name
Names the security policy. This is a one-part name. The name must not identify an existing security policy at the current server (SQLSTATE 42710).

COMPONENTS *component-name*,...
Identifies a security label component. The name must identify a security label component that already exists at the current server (SQLSTATE 42704). The same security component must not be specified more than once for the security policy (SQLSTATE 42713). No more than 16 security label components can be specified for a security policy (SQLSTATE 54062).

WITH DB2LBACRULES
Indicates what rule set that will be used when comparing security labels that are part of this security policy. There is currently only one rule set: DB2LBACRULES.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL or RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
Specifies the action that is to be taken when a user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement issued against a table that is protected with this security policy. A user's security label and exemption credentials determine the user's authorization to write an explicitly provided security label. The default is OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL
Indicates that the value of the user's security label, rather than the explicitly specified security label, is to be used for write access during an insert or update operation.

RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
Indicates that the insert or update operation will fail if the user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement (SQLSTATE 42519).

Notes

- ***DB2LBACRULES rule set:*** DB2LBACRULES is a predefined set of rules that includes the following rules: DB2LBACREADARRAY, DB2LBACREADSET, DB2LBACREADTREE, DB2LBACWRITEARRAY, DB2LBACWRITESET, DB2LBACWRITETREE.
- Group and role authorizations are not considered by default when a security policy is created. Use the ALTER SECURITY POLICY statement to change this behavior and have them considered.

Examples

- *Example 1:* Create a security policy named DATA_ACCESS that uses the DB2LBACRULES rule set and has two components: LEVEL and COMPARTMENTS, in that order. Assume that both components already exist.

```
CREATE SECURITY POLICY DATA_ACCESS
  COMPONENTS LEVEL, COMPARTMENTS
  WITH DB2LBACRULES
```

- *Example 2:* Create a security policy named CONTRIBUTIONS that has the components MEMBER and BADGE, which are assumed to already exist.

```
CREATE SECURITY POLICY CONTRIBUTIONS
  COMPONENTS MEMBER, BADGE
  WITH DB2LBACRULES
```

CREATE SEQUENCE

The CREATE SEQUENCE statement defines a sequence at the application server.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

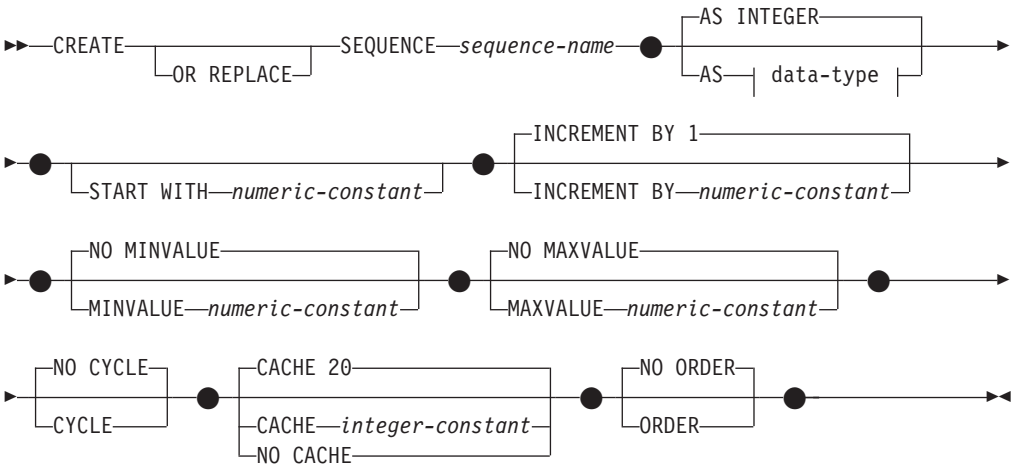
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the sequence does not exist
- CREATEIN privilege on the schema, if the schema name of the sequence refers to an existing schema
- DBADM authority

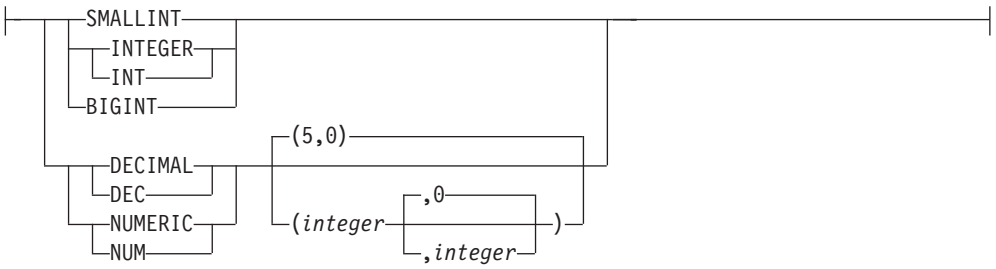
To replace an existing sequence, the authorization ID of the statement must be the owner of the existing sequence (SQLSTATE 42501).

Syntax



data-type:

built-in-type:



Notes:

1 The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type.

Description

OR REPLACE
Specifies to replace the definition for the sequence if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the sequence are not affected. This option is ignored if a definition for the sequence does not exist at the current server. This option can be specified only by the owner of the object.

sequence-name
Names the sequence. The combination of name, and the implicit or explicit schema name must not identify an existing sequence at the current server (SQLSTATE 42710).

The unqualified form of sequence-name is an SQL identifier. The qualified form is a qualifier followed by a period and an SQL identifier. The qualifier is a schema name.

If the sequence name is explicitly qualified with a schema name, the schema name cannot begin with 'SYS' or an error (SQLSTATE 42939) is raised.

AS *data-type*
Specifies the data type to be used for the sequence value. The data type can be any exact numeric type (SMALLINT, INTEGER, BIGINT or DECIMAL) with a scale of zero, or a user-defined distinct type or reference type for which the source type is an exact numeric type with a scale of zero (SQLSTATE 42815). The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type (SQLSTATE 428H2). The default is INTEGER.

START WITH *numeric-constant*
Specifies the first value for the sequence. This value can be any positive or negative value that could be assigned to a column of the data type associated with the sequence (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA). The default is MINVALUE for ascending sequences and MAXVALUE for descending sequences.

This value is not necessarily the value that a sequence would cycle to after reaching the maximum or minimum value of the sequence. The START WITH clause can be used to start a sequence outside the range that is used for cycles. The range used for cycles is defined by MINVALUE and MAXVALUE.

INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the sequence. This value can be any positive or negative value that could be assigned to a column of the

data type associated with the sequence (SQLSTATE 42815). The value must not exceed the value of a large integer constant (SQLSTATE 42820) and must not contain nonzero digits to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence. If this value is 0 or positive, this is an ascending sequence. The default is 1.

MINVALUE or NO MINVALUE
Specifies the minimum value at which a descending sequence either cycles or stops generating values, or an ascending sequence cycles to after reaching the maximum value.

MINVALUE *numeric-constant*
Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to a column of the data type associated with the sequence (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

NO MINVALUE
For an ascending sequence, the value is the START WITH value, or 1 if START WITH is not specified. For a descending sequence, the value is the minimum value of the data type associated with the sequence. This is the default.

MAXVALUE or NO MAXVALUE
Specifies the maximum value at which an ascending sequence either cycles or stops generating values, or a descending sequence cycles to after reaching the minimum value.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to a column of the data type associated with the sequence (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

NO MAXVALUE
For an ascending sequence, the value is the maximum value of the data type associated with the sequence. For a descending sequence, the value is the START WITH value, or -1 if START WITH is not specified.

CYCLE or NO CYCLE
Specifies whether the sequence should continue to generate values after reaching either its maximum or minimum value. The boundary of the sequence can be reached either with the next value landing exactly on the boundary condition, or by overshooting it.

CYCLE
Specifies that values continue to be generated for this sequence after the maximum or minimum value has been reached. If this option is used, after an ascending sequence reaches its maximum value it generates its minimum value; after a descending sequence reaches its minimum value it generates its maximum value. The maximum and minimum values for the sequence determine the range that is used for cycling.

When CYCLE is in effect, then duplicate values can be generated for the sequence.

NO CYCLE
Specifies that values will not be generated for the sequence once the maximum or minimum value for the sequence has been reached. This is the default.

CACHE or NO CACHE
Specifies whether to keep some preallocated values in memory for faster access. This is a performance and tuning option.

CACHE *integer-constant*
Specifies the maximum number of sequence values that are preallocated and kept in memory. Preallocating and storing values in the cache reduces synchronous I/O to the log when values are generated for the sequence.

In the event of a system failure, all cached sequence values that have not been used in committed statements are lost (that is, they will never be used). The value specified for the CACHE option is the maximum number of sequence values that could be lost in case of system failure.

The minimum value is 2 (SQLSTATE 42815). The default value is CACHE 20.

In a multi-partition or DB2 pureScale environment, use the CACHE and NO ORDER options to allow multiple DB2 members to cache sequence values simultaneously.

In a DB2 pureScale environment, if both CACHE and ORDER are specified, the specification of ORDER overrides the specification of CACHE and instead NO CACHE will be in effect.

NO CACHE
Specifies that values of the sequence are not to be preallocated. It ensures that there is not a loss of values in the case of a system failure, shutdown or database deactivation. When this option is specified, the values of the sequence are not stored in the cache. In this case, every request for a new value for the sequence results in synchronous I/O to the log.

NO ORDER or ORDER
Specifies whether the sequence numbers must be generated in order of request.

ORDER
Specifies that the sequence numbers are generated in order of request.

NO ORDER
Specifies that the sequence numbers do not need to be generated in order of request. This is the default.

Notes

- It is possible to define a constant sequence, that is, one that would always return a constant value. This could be done by specifying an INCREMENT value of zero and a START WITH value that does not exceed MAXVALUE, or by specifying the same value for START WITH, MINVALUE and MAXVALUE. For a constant sequence, each time NEXT VALUE is invoked for the sequence, the same value is returned. A constant sequence can be used as a numeric global variable. ALTER SEQUENCE can be used to adjust the values that will be generated for a constant sequence.
- A sequence can be cycled manually by using the ALTER SEQUENCE statement. If NO CYCLE is implicitly or explicitly specified, the sequence can be restarted

or extended using the ALTER SEQUENCE statement to cause values to continue to be generated once the maximum or minimum value for the sequence has been reached.

- A sequence can be explicitly defined to cycle by specifying the CYCLE keyword. Use the CYCLE option when defining a sequence to indicate that the generated values should cycle once the boundary is reached. When a sequence is defined to automatically cycle (that is, CYCLE was explicitly specified), the maximum or minimum value generated for a sequence might not be the actual MAXVALUE or MINVALUE specified, if the increment is a value other than 1 or -1. For example, the sequence defined with `START WITH=1, INCREMENT=2, MAXVALUE=10` will generate a maximum value of 9, and will not generate the value 10. When defining a sequence with CYCLE, carefully consider the impact of the values for MINVALUE, MAXVALUE and START WITH.
- Caching sequence numbers implies that a range of sequence numbers can be kept in memory for fast access. When an application accesses a sequence that can allocate the next sequence number from the cache, the sequence number allocation can happen quickly. However, if an application accesses a sequence that cannot allocate the next sequence number from the cache, the sequence number allocation may require having to wait for I/O operations to persistent storage. The choice of the value for CACHE should be done keeping in mind the performance and application requirements tradeoffs.
- The definer of a sequences is granted ALTER and USAGE privileges with the grant option. The owner of the sequence can drop the sequence.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - A comma can be used to separate multiple sequence options
  - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER can be specified in place of NO MINVALUE, NO MAXVALUE, NO CYCLE, NO CACHE, and NO ORDER, respectively
- ***Considerations for a multi-partition or DB2 pureScale environment***:
  - In a multi-partition or DB2 pureScale environment, if the CACHE and NO ORDER options are in effect, multiple caches can be active simultaneously and the requests for next value assignments from different members might not result in the assignment of values in strict numeric order. Assume, for example, that members DB2A and DB2B are using the same sequence, and DB2A gets the cache values 1 to 20 and DB2B gets the cache values 21 to 40. In this scenario, if DB2A requested the next value first, then DB2B requested, and then DB2A requested again, the actual order of values assigned would be 1,21,2. Therefore, to guarantee that sequence numbers are generated in strict numeric order among multiple members using the same sequence concurrently, specify the ORDER option.
  - In a DB2 pureScale environment, using the ORDER or NO CACHE option ensures that the values assigned to a sequence which is shared by applications across multiple members are in strict numeric order. If ORDER is specified, then NO CACHE is implied even if CACHE *n* is specified

Example

Create a sequence called ORG_SEQ that starts at 1, increments by 1, does not cycle, and caches 24 values at a time:

```
CREATE SEQUENCE ORG_SEQ
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO CYCLE
  CACHE 24
```

CREATE SERVICE CLASS

The CREATE SERVICE CLASS statement defines a service class.

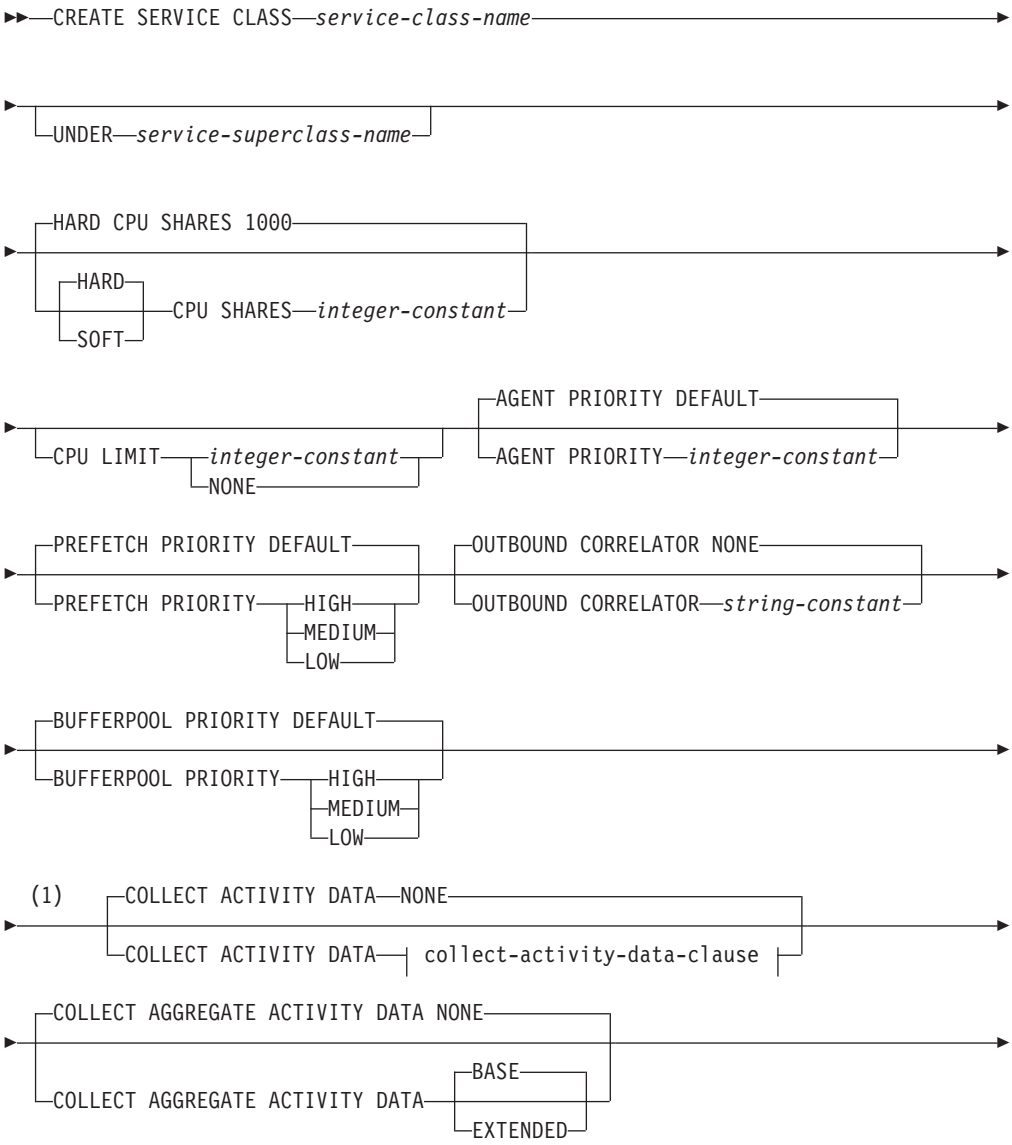
Invocation

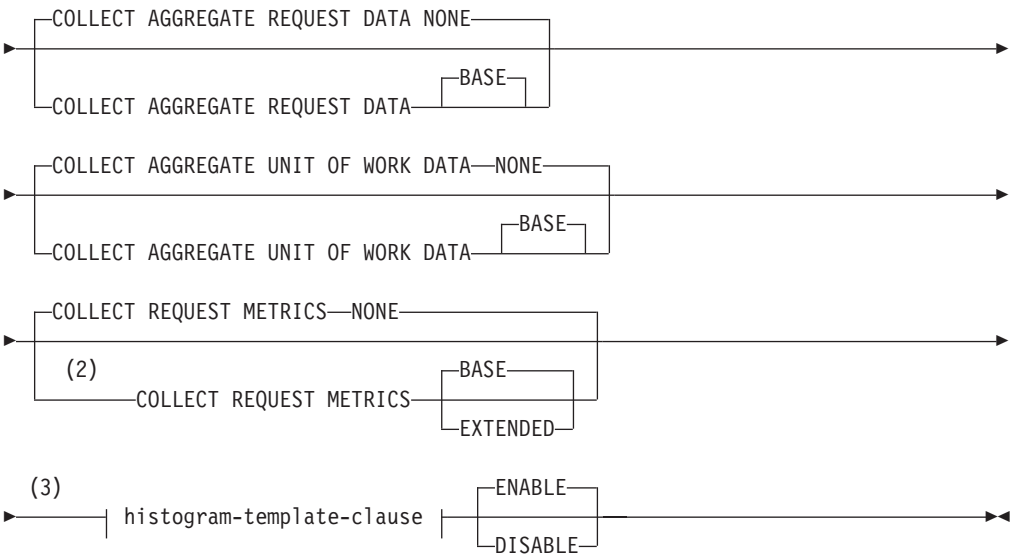
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

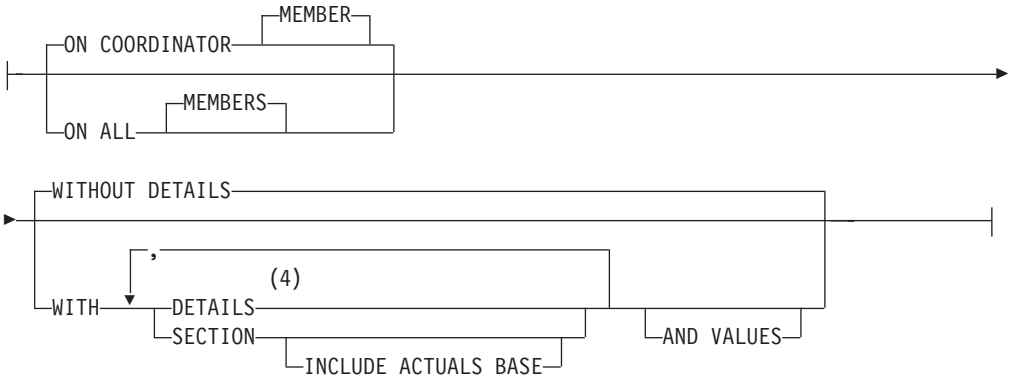
The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

Syntax

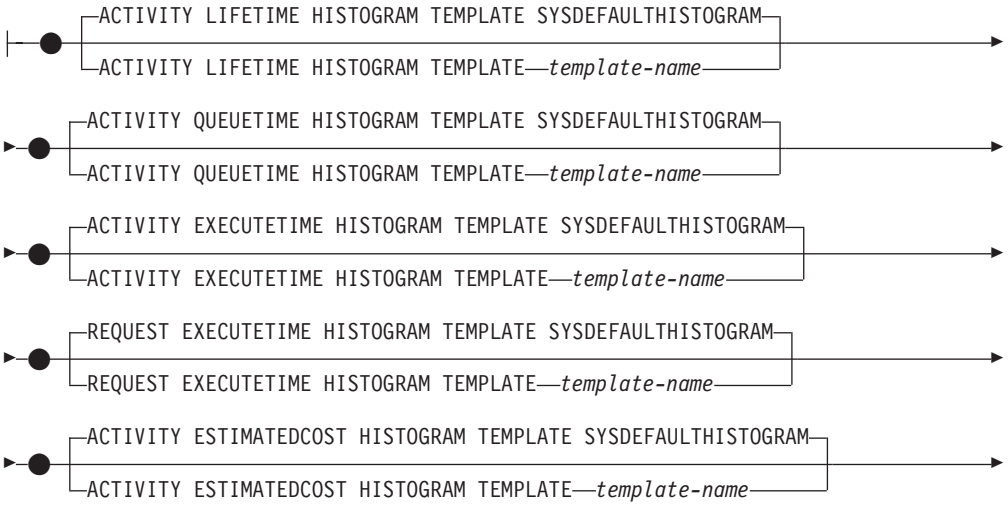


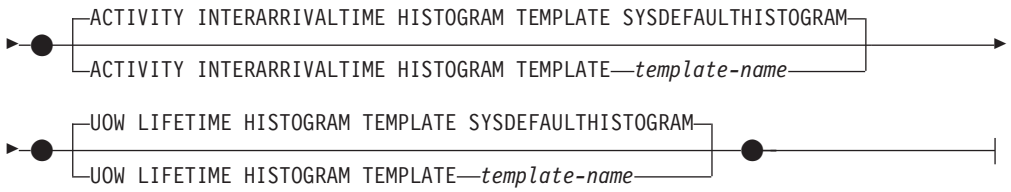


collect-activity-data-clause:



histogram-template-clause:





Notes:

1 All COLLECT clauses except for COLLECT REQUEST METRICS are valid only for a service subclass.

2 The COLLECT REQUEST METRICS clause is valid only for a service superclass.

3 The HISTOGRAM TEMPLATE clauses are valid only for a service subclass.

4 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

Description

service-class-name
Names the service class. This is a one-part name. It is an SQL identifier (either ordinary or delimited). If the service class is a service superclass, the *service-class-name* must not identify a service superclass that already exists in the catalog (SQLSTATE 42710). If the service class is a service subclass, the *service-class-name* must not identify a service subclass that already exists under the service superclass (SQLSTATE 42710). If the service class is a service subclass, the *service-class-name* must not be the same as its service superclass (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

UNDER *service-superclass-name*
Specifies that the service class is a subclass of service superclass *service-superclass-name*. If UNDER is not specified, the service class is a service superclass. The *service-superclass-name* must identify a service superclass that exists for the database (SQLSTATE 42704). The service superclass cannot be a default service class (SQLSTATE 5U029).

HARD CPU SHARES 1000 or HARD CPU SHARES *integer-constant* **or SOFT CPU SHARES** *integer-constant*
Specifies the number of shares of CPU resources that the WLM dispatcher allocates to this service class when work is executing within this service class. Valid values for the *integer-constant* are integers between 1 and 65535. The default is **HARD CPU SHARES 1000**, which allocates 1000 hard shares to this service class. Qualifying **CPU SHARES** with the keyword **HARD**, or specifying **CPU SHARES** without qualifying it with the keyword **HARD** or **SOFT**, indicates that hard CPU shares are to be allocated to this service class. Specifying the keyword **SOFT** indicates that soft CPU shares are to be allocated to this service class. To use hard and soft CPU shares with DB2 workload manager dispatcher, you must enable the **wlm_disp_cpu_shares** database manager configuration parameter.

CPU LIMIT *integer-constant* **or CPU LIMIT NONE**
Specifies the maximum percentage of the CPU resources that the WLM dispatcher can assign to this service class. Valid values for the *integer-constant* are integers between 1 and 100. You can also specify **CPU LIMIT NONE** to indicate that there is no CPU limit.

AGENT PRIORITY DEFAULT or AGENT PRIORITY *integer-constant*
Specifies the relative (delta) operating system priority of agents running in the service class or the normal priority of threads running in DB2. The default value is DEFAULT.

Important: The **agentpri** database manager configuration is deprecated since Version 9.5. It can still be used in pre-Version 9.5 data servers and clients. Also, agent priority for the WLM service class has been deprecated in Version 10.1 and might be removed in a future release. Start to use the WLM dispatcher capability instead of agent priority. For more information, see "Agent priority of service classes has been deprecated" in *What's New for DB2 Version 10.1*.

When set to DEFAULT, no special action is taken, and agents in the service class are scheduled according to the normal priority that the operating system schedules all DB2 threads. When this parameter is set to a value other than DEFAULT, agents are set to a priority that is equal to the normal priority plus AGENT PRIORITY when the next activity begins. For example, if the normal priority is 20 and AGENT PRIORITY is set to -10, the priority of agents in the service class is set to 20 - 10 = 10.

Note: Agent priority and WLM dispatcher shares cannot be used together. When the dispatcher is enabled by setting the value of the **wlm_dispatcher** database manager configuration parameter to ON, the specified agent priority setting is ignored and agent priority is set to the default value until the dispatcher is disabled.

DB2 workload manager (WLM) does not assign service class agent priority to work being done within a fenced mode process (FMP). Fenced procedures do not run their logic within a service class. These fenced procedures run within the DB2 FMP and this work is not done by DB2 agents. As a reminder, DB2 WLM controls DB2 agents.

On UNIX operating systems and Linux, valid values are DEFAULT and -20 to 20 (SQLSTATE 42615). Negative values denote a higher relative priority. Positive values denote a lower relative priority.

On Windows operating systems, valid values are DEFAULT and -6 to 6 (SQLSTATE 42615). Negative values denote a lower relative priority. Positive values denote a higher relative priority.

If AGENT PRIORITY is DEFAULT for a service subclass, it inherits the AGENT PRIORITY value of its parent superclass. AGENT PRIORITY cannot be altered for a default subclass (SQLSTATE 5U032). AGENT PRIORITY must be set to DEFAULT if OUTBOUND CORRELATOR is set (SQLSTATE 42613).

Note: On AIX, the instance owner must have CAP_NUMA_ATTACH and CAP_PROPAGATE capabilities to set a higher relative priority for agents in a service class using AGENT PRIORITY. To grant these capabilities, logon as root and run the following command:

```
chuser capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE
```

On Solaris 10 or higher, the instance owner must have the proc_priocntl privilege to set a higher relative priority for agents in a service class using AGENT PRIORITY. To grant this privilege, logon as `root` and run the following command:

```
usermod -K defaultpriv=basic,proc_priocntl db2user
```

In this example, proc_priocntl is added to the default privilege set of user db2user.

Moreover, when DB2 is running in a non-global zone of Solaris, the proc_priocntl privilege must be added to the zone's limit privilege set. To grant this privilege to the zone, logon as `root` and run the following command:

```
global# zonecfg -z db2zone
zonecfg:db2zone> set limitpriv="default,proc_priocntl"
```

In this example, proc_priocntl is added to the limit privilege set of zone db2zone.

On Solaris 9, there is no facility for DB2 to raise the relative priority of agents. Upgrade to Solaris 10 or higher to use the service class agent priority.

PREFETCH PRIORITY DEFAULT | HIGH | MEDIUM | LOW
This parameter controls the priority with which agents in the service class can submit their prefetch requests. Valid values are HIGH, MEDIUM, LOW, or DEFAULT (SQLSTATE 42615). HIGH, MEDIUM, and LOW mean that prefetch requests will be submitted to the high, medium, and low priority queues, respectively. Prefetchers empty the priority queue in order from high to low. Agents in the service class submit their prefetch requests at the PREFETCH PRIORITY level when the next activity begins. If PREFETCH PRIORITY is altered after a prefetch request is submitted, the request priority does not change. The default value is DEFAULT, which is internally mapped to MEDIUM for service superclasses. If DEFAULT is specified for a service subclass, it inherits the PREFETCH PRIORITY of its parent superclass.

PREFETCH PRIORITY cannot be altered for a default subclass (SQLSTATE 5U032).

OUTBOUND CORRELATOR NONE or OUTBOUND CORRELATOR *string-constant*
Specifies whether or not to associate threads from this service class to an external workload manager service class.

If OUTBOUND CORRELATOR is set to a *string-constant* for the service superclass and OUTBOUND CORRELATOR NONE is set for a service subclass, the service subclass inherits the OUTBOUND CORRELATOR of its parent. OUTBOUND CORRELATOR must be set to NONE if the AGENT PRIORITY is not set to DEFAULT (SQLSTATE 42613). The default is OUTBOUND CORRELATOR NONE.

OUTBOUND CORRELATOR NONE
For a service superclass, specifies that there is no external workload manager service class association with this service class, and for a service subclass, specifies that the external workload manager service class association is the same as its parent.

OUTBOUND CORRELATOR *string-constant*
Specifies the *string-constant* that is to be used as a correlator to associate threads from this service class to an external workload manager service class. The external workload manager must be active (SQLSTATE 5U030). The external workload manager should be set up to recognize the value of *string-constant*.

BUFFERPOOL PRIORITY DEFAULT | HIGH | MEDIUM | LOW
This parameter controls the bufferpool priority of pages fetched by activities in this service class. Valid values are HIGH, MEDIUM, LOW or DEFAULT (SQLSTATE 42615). Pages fetched by activities in a service class with higher bufferpool priority are less likely to be swapped out than pages fetched by activities in a service class with lower bufferpool priority. The default value is

DEFAULT, which is internally mapped to LOW for service superclasses. If DEFAULT is specified for a service subclass, it inherits the BUFFERPOOL PRIORITY from its parent superclass.

BUFFERPOOL PRIORITY cannot be altered for a default subclass (SQLSTATE 5U032).

COLLECT ACTIVITY DATA
Specifies that information about each activity that executes in this service class is to be sent to any active activities event monitor when the activity completes. The default is COLLECT ACTIVITY DATA NONE. The COLLECT ACTIVITY DATA clause is valid only for a service subclass.

NONE
Specifies that activity data should not be collected for each activity that executes in this service class.

ON COORDINATOR MEMBER
Specifies that activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that activity data is to be collected at all members where the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. If the AND VALUES clause is specified, activity input values will be collected only for the members of the coordinator.

WITHOUT DETAILS
Specifies that data about each activity that executes in the service class is to be sent to any active activities event monitor, when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. Section actuals will be collected on any member where the activity data is collected.

INCLUDE ACTUALS BASE
Specifies that section actuals should also be collected on any partition where the activity data is collected. For section actuals to be collected, either INCLUDE ACTUALS clause must be specified or the **section_actuals** database configuration parameter must be set.

The effective setting for the collection of section actuals is the combination of the INCLUDE ACTUALS clause, the **section_actuals** database configuration parameter, and the <collectsectionactuals> setting specified on the WLM_SET_CONN_ENV routine. For example, if INCLUDE ACTUALS BASE is specified, yet the **section_actuals** database

configuration parameter value is NONE and <collectsectionactuals> is set to NONE, then the effective setting for the collection of section actuals is BASE.

BASE specifies that the following should be enabled and collected during the activity's execution:

- Basic operator cardinality counts
- Statistics for each object referenced (DML statements only)

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

COLLECT AGGREGATE ACTIVITY DATA
Specifies that aggregate activity data should be captured for this service class and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default when COLLECT AGGREGATE ACTIVITY DATA is not specified is COLLECT AGGREGATE ACTIVITY DATA NONE. The default when COLLECT AGGREGATE ACTIVITY DATA is specified is COLLECT AGGREGATE ACTIVITY DATA BASE. The COLLECT AGGREGATE ACTIVITY DATA clause is valid only for a service subclass.

BASE
Specifies that basic aggregate activity data should be captured for this service class and sent to the statistics event monitor, if one is active. Basic aggregate activity data includes:

- Estimated activity cost high watermark
- Rows returned high watermark
- Temporary table space usage high watermark

  Note: Only activities that have an SQLTEMPSPACE threshold applied to them participate in this high watermark.
- Activity life time histogram
- Activity queue time histogram
- Activity execution time histogram

EXTENDED
Specifies that all aggregate activity data should be captured for this service class and sent to the statistics event monitor, if one is active. This includes all basic aggregate activity data plus:

- Activity data manipulation language (DML) estimated cost histogram
- Activity DML inter-arrival time histogram

NONE
Specifies that no aggregate activity data should be captured for this service class.

COLLECT AGGREGATE REQUEST DATA
Specifies that aggregate request data should be captured for this service class and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval specified by the **wlm_collect_int** database configuration parameter. The default is COLLECT AGGREGATE REQUEST DATA NONE. The COLLECT AGGREGATE REQUEST DATA clause is valid only for a service subclass.

BASE
Specifies that basic aggregate request data should be captured for this service class and sent to the statistics event monitor, if one is active.

NONE
Specifies that no aggregate request data should be captured for this service class.

COLLECT AGGREGATE UNIT OF WORK DATA
Specifies that aggregate unit of work data is to be captured for this service class and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval specified by the **wlm_collect_int** database configuration parameter. The default is COLLECT AGGREGATE UNIT OF WORK DATA NONE. The COLLECT AGGREGATE UNIT OF WORK DATA clause is valid only for a service subclass.

BASE
Specifies that basic aggregate unit of work data is to be captured for this service class and sent to the statistics event monitor, if one is active. Basic aggregate unit of work includes:

- Unit of work lifetime histogram

NONE
Specifies that no aggregate unit of work data is to be collected for this service class.

COLLECT REQUEST METRICS
Specifies that monitor metrics should be collected for any request submitted by a connection that is associated with the specified service superclass and sent to the statistics and unit of work event monitors, if active. The default is COLLECT REQUEST METRICS NONE. The COLLECT REQUEST METRICS clause is valid only for a service superclass (SQLSTATE 50U44).

Note: The effective request metrics collection setting is the combination of the attribute specified by the COLLECT REQUEST METRICS clause on the service superclass associated with the connection submitting the request, and the **mon_req_metrics** database configuration parameter. If either the service superclass attribute or the configuration parameter has a value other than NONE, metrics will be collected for the request.

NONE
Specifies that no metrics will be collected for any request submitted by a connection associated with the service superclass.

BASE
Specifies that basic metrics will be collected for any request submitted by a connection associated with the service superclass.

EXTENDED
Specifies that basic aggregate request data should be captured for this service class and sent to the statistics event monitor, if one is active. In addition, specifies that the values for the following monitor elements should be determined with additional granularity:

- **total_section_time**
- **total_section_proc_time**
- **total_routine_user_code_time**
- **total_routine_user_code_proc_time**
- **total_routine_time**

histogram-template-clause
Specifies the histogram templates to use when collecting aggregate activity data for activities executing in the service class. The HISTOGRAM TEMPLATE clause is valid only for a service subclass.

ACTIVITY LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of DB2 activities running in the service class during a specific interval. This time includes both time queued and time executing. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY QUEUETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the service class are queued during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the service class are executing during a specific interval. This time does not include the time spent queued. Activity execution time is collected in this histogram at the coordinator member only. The time does not include idle time. Idle time is the time between the execution of requests belonging to the same activity when no work is being done. An example of idle time is the time between the end of opening a cursor and the start of fetching from that cursor. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option. Only activities at nesting level 0 are considered for inclusion in the histogram.

REQUEST EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 requests running in the service class are executing during a specific interval. This time does not include the time spent queued. Request execution time is collected in this histogram on each member where the request executes. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE REQUEST DATA clause is specified with the BASE option.

ACTIVITY ESTIMATEDCOST HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the estimated cost, in timerons, of DML activities running in the service class. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option. Only activities at nesting level 0 are considered for inclusion in the histogram.

ACTIVITY INTERARRIVALTIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, between the arrival of one DML activity and the arrival of the next DML activity. The default is

SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

UOW LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of units of work running in the service class during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE UNIT OF WORK DATA clause is specified with the BASE option.

ENABLE or DISABLE
Specifies whether or not connections and activities can be mapped to the service class. The default is ENABLE.

ENABLE
Connections and activities can be mapped to the service class.

DISABLE
Connections and activities cannot be mapped to the service class. New connections or activities that are mapped to a disabled service class will be rejected (SQLSTATE 5U028). When a service superclass is disabled, its service subclasses are also disabled. When the service superclass is re-enabled, its service subclasses return to states that are defined in the system catalog. A default service class cannot be disabled (SQLSTATE 5U032).

Rules

- The maximum number of service subclasses that can be created under a service superclass is 61 (SQLSTATE 5U027).
- The maximum number of service superclasses that can be created for a database is 64 (SQLSTATE 5U027).
- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U027). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (histogram template)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (service class)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (threshold)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (work action set)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (work class set)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (workload)
  - GRANT (workload privileges) or REVOKE (workload privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- A default subclass, SYSDEFAULTSUBCLASS, is automatically created for every service superclass.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all members. If an uncommitted WLM-exclusive SQL statement is

executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.

- Changes are written to the system catalog, but do not take effect until after a COMMIT statement, even for the connection that issues the statement.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

Examples

- *Example 1:* Create a service superclass named PETSALES. The default subclass for PETSALES is automatically created.

```
CREATE SERVICE CLASS PETSALES
```

- *Example 2:* Create a service subclass named DOGSALES under service superclass PETSALES. Set service class DOGSALES as disabled.

```
CREATE SERVICE CLASS DOGSALES UNDER PETSALES DISABLE
```

- *Example 3:* Create a service superclass named BARNSALES with a prefetcher priority of LOW. The default subclass for BARNSALES is automatically created. Prefetch requests submitted by agents in the BARNSALES service class will go to the low priority prefetch queue.

```
CREATE SERVICE CLASS BARNSALES PREFETCH PRIORITY LOW
```

CREATE SERVER

The CREATE SERVER statement defines a data source to a federated database.

In this statement, the term SERVER and the parameter names that start with *server-* refer only to data sources in a federated system. They do not refer to the federated server in such a system, or to DRDA application servers.

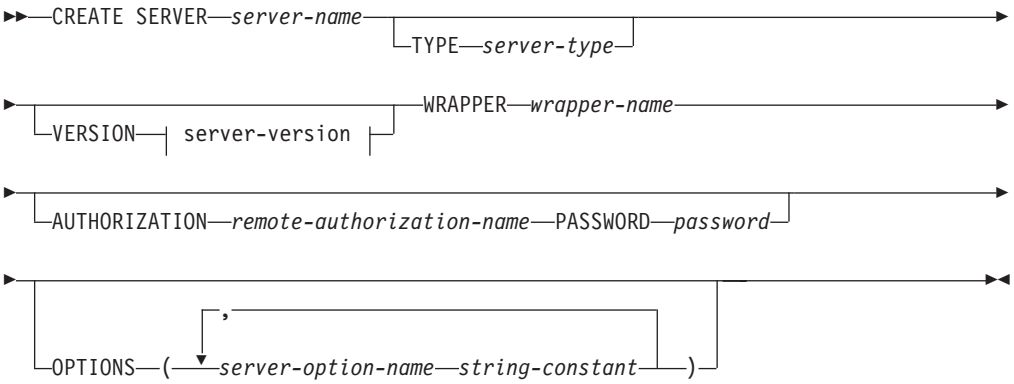
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

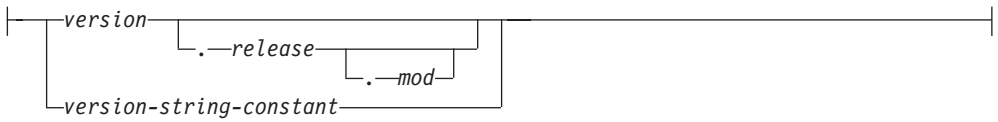
Authorization

The privileges held by the authorization ID of the statement must include DBADM authority.

Syntax



server-version:



Description

server-name

Names the data source that is being defined to the federated database. The name must not identify a data source that is described in the catalog. The *server-name* must not be the same as the name of any table space in the federated database.

A server definition for relational data sources usually represents a remote database. Some relational database management systems, such as Oracle, do not allow multiple databases within each instance. Instead, each instance represents a server within a federated system.

For nonrelational data sources, the purpose of a server definition varies from data source to data source. Some server definitions map to a search type and daemon, a website, or a web server. For other nonrelational data sources, a server definition is created because the hierarchy of federated objects requires that data source files (identified by nicknames) are associated with a specific server object.

TYPE *server-type*
Specifies the type of data source denoted by *server-name*. This parameter is required by some wrappers.

VERSION
Specifies the version of the data source denoted by *server-name*. This parameter is required by some wrappers.

version
Specifies the version number. The value must be an integer.

release
Specifies the number of the release of the version denoted by *version*. The value must be an integer.

mod
Specifies the number of the modification of the release denoted by *release*. The value must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release* and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*
Names the wrapper that the federated server uses to interact with the server object specified by *server-name*.

AUTHORIZATION *remote-authorization-name*
Required only for DB2 family data sources. Specifies the authorization ID under which any necessary actions are performed at the data source when the CREATE SERVER statement is processed. This authorization ID is not used when establishing subsequent connections to the server.

This ID must hold the authority (BINDADD or its equivalent) that the necessary actions require. If the *remote-authorization-name* is specified in mixed or lowercase characters (and the remote data source has case sensitive authorization names), the *remote-authorization-name* should be enclosed by double quotation marks.

PASSWORD *password*
Required only for DB2 family data sources. Specifies the password associated with the authorization ID represented by *remote-authorization-name*. If the *password* is specified in mixed or lowercase characters (and the remote data source has case sensitive passwords), the *password* should be enclosed by double quotation marks.

OPTIONS
Indicates the options that are enabled when the server definition is created. Server options are used to configure the server definition. Some server options can be used to create the server definition for any data source. Some server options are specific to a particular data source.

server-option-name
Names a server option that will be used to either configure or provide information about the data source denoted by *server-name*.

string-constant
Specifies the setting for *server-option-name* as a character string constant.

Notes

- The *password* should be specified when the data source requires a password. If any letters in *password* must be in lowercase, enclose *password* in quotation marks.
- If the CREATE SERVER statement is used to define a DB2 family instance as a data source, DB2 may need to bind certain packages to that instance. If binding is required, the *remote-authorization-name* in the statement must have BIND authority. The time required for the bind operation to complete is dependent on data source speed and network connection speed.
- DB2 does not verify that the specified server version matches the remote server version. Specifying an incorrect server version can result in SQL errors when you access nicknames that belong to the DB2 server definition. This is most likely when you specify a server version that is later than the remote server version. In that case, when you access nicknames that belong to the server definition, DB2 might send SQL that the remote server does not recognize.
- **Syntax alternatives:** The following syntax is supported for compatibility with previous versions of DB2:
  - ADD can be specified before *server-option-name string-constant*.

Examples

- *Example 1:* Register a server definition to access a DB2 for z/OS and OS/390®, Version 7.1 data source. CRANDALL is the name assigned to the DB2 for z/OS and OS/390 server definition. DRDA is the name of the wrapper used to access this data source. In addition, specify that:
  - GERALD and drowssap are the authorization ID and password under which packages are bound at CRANDALL when this statement is processed.
  - The alias for the DB2 for z/OS and OS/390 database that was specified with the CATALOG DATABASE statement is CLIENTS390.
  - The authorization IDs and passwords under which CRANDALL can be accessed are to be sent to CRANDALL in uppercase.
  - CLIENTS390 and the federated database use the same collating sequence.

```
CREATE SERVER CRANDALL
  TYPE DB2/ZOS
  VERSION 7.1
  WRAPPER DRDA
  AUTHORIZATION "GERALD"
  PASSWORD drowssap
  OPTIONS
      (DBNAME 'CLIENTS390',
      FOLD_ID 'U',
      FOLD_PW 'U',
      COLLATING_SEQUENCE 'Y')
```

- *Example 2:* Register a server definition to access an Oracle 9 data source. CUSTOMERS is the name assigned to the Oracle server definition. NET8 is the name of the wrapper used to access this data source. In addition, specify that:
  - ABC is the name of the node where the Oracle database server resides.
  - The CPU for the federated server runs twice as fast as the CPU that supports CUSTOMERS.

  - The I/O devices at the federated server process data one and a half times as fast as the I/O devices at CUSTOMERS.

```
CREATE SERVER CUSTOMERS
  TYPE ORACLE
  VERSION 9
  WRAPPER NET8
  OPTIONS
      (NODE 'ABC',
      CPU_RATIO '2.0',
      IO_RATIO '1.5')
```

- *Example 3*: Register a server definition for the Excel wrapper. The server definition is required to preserve the hierarchy of federated objects. BIOCHEM_LAB is the name assigned to the Excel server definition. EXCEL_2000_WRAPPER is the name of the wrapper used to access this data source.

```
CREATE SERVER BIOCHEM_DATA
  WRAPPER EXCEL_2000_WRAPPER
```

CREATE STOGROUP

The CREATE STOGROUP statement defines a new storage group within the database, assigns storage paths to the storage group, and records the storage group definition and attributes in the catalog.

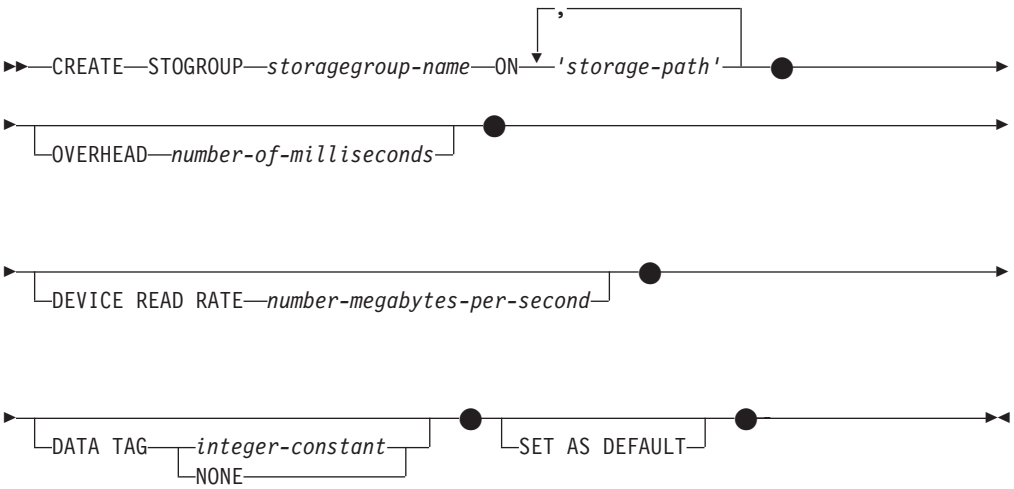
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax



Description

storagegroup-name
Names the storage group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *storagegroup-name* must not identify a storage group that already exists at the current server (SQLSTATE 42710). The *storagegroup-name* must not begin with the characters 'SYS' (SQLSTATE 42939).

ON Specifies storage paths to be added for the named storage group. For partitioned database environments, the same storage paths will be defined on all database partitions unless database partition expressions are used.

storage-path
A string constant that specifies either an absolute path or the letter name of a drive (Windows operating systems only) on which containers for automatic storage table spaces are to be created. The string can include database partition expressions to specify database partition number information in the storage path.

The maximum length of a path is 175 characters (SQLSTATE 54036).A storage path being added must be valid according to the naming rules for paths, and must be accessible (SQLSTATE 57019). Similarly, in a partitioned database environment, the storage path must exist and be accessible on every database partition (SQLSTATE 57019).

OVERHEAD *number-of-milliseconds*
Specifies the I/O controller usage and disk seek and latency time. This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all storage paths, set the value to a numeric literal which represents the average for all storage paths that belong to the storage group.

If the OVERHEAD clause is not specified, the OVERHEAD will be set to 6.725 milliseconds.

DEVICE READ RATE *number-megabytes-per-second*
Specifies the device specification for the read transfer rate in megabytes per second. This value is used to determine the cost of I/O during query optimization. The value of *number-megabytes-per-second* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all storage paths, set the value to a numeric literal which represents the average for all storage paths that belong to the storage group.

If the DEVICE READ RATE clause is not specified, the DEVICE READ RATE will be set to the built-in default of 100 megabytes per second.

DATA TAG *integer-constant* **or DATA TAG NONE**
Specifies a tag for the data for table spaces using this storage group unless explicitly overridden by the table space definition. This value can be used as part of a WLM configuration in a work class definition or referenced within a threshold definition. For more information, see the CREATE WORK CLASS SET and CREATE THRESHOLD statements.

integer-constant
Valid values for *integer-constant* are integers from 1 to 9.

NONE
If NONE is specified, there is no data tag.

SET AS DEFAULT
Specifies the storage group being created is designated as the default storage group. If there is no default storage group, the first one created will be designated the default even if this clause is not specified. Since there can only be one storage group designated as the default storage group, specifying this clause removes the default attribute from the existing default storage group. Specifying a new default storage group has no affect to the storage group used by existing table spaces.

Rules

- The CREATE STOGROUP statement cannot be executed while a database partition server is being added (SQLSTATE 55071).
- A storage group can have up to 128 defined storage paths (SQLSTATE 5U009).
- A database instance can have up to 256 defined storage groups (SQLSTATE 54035).

Notes

- ***Calculation of free space:*** When free space is calculated for a storage path on a database partition, the database manager checks for the existence of the following directories or mount points within the storage path, and will use the first one that is found.

  ```
  <storage path>/<instance name>/NODE####/<database name>
  <storage path>/<instance name>/NODE####
  <storage path>/<instance name>
  <storage path>
  ```

  Where:

  – `<storage path>` is a storage path associated with the database.
  – `<instance name>` is the instance under which the database resides.
  – `NODE####` corresponds to the database partition number (for example, NODE0000 or NODE0001).
  – `<database name>` is the name of the database.

- ***Isolating multiple database partitions under one storage path:*** File systems can be mounted at a point beneath the storage path, and the database manager will recognize that the actual amount of free space available for table space containers might not be the same amount that is associated with the storage path directory itself.

  Consider an example in which two logical database partitions exist on one physical computer, and there is a single storage path (`/db2data`). Each database partition will use this storage path, but you might want to isolate the data from each partition within its own file system. In this case, a separate file system can be created for each partition and it can be mounted at `/db2data/<instance>/NODE####`. When creating containers on the storage path and determining free space, the database manager will not retrieve free space information for `/db2data`, but instead will retrieve it for the corresponding `/db2data/<instance>/NODE####` directory.

- ***Multiple storage paths:*** A storage path can be added to different storage groups, or to the same storage group multiple times.
- ***Similar media characteristics:*** Ensure that the storage paths added to a storage group have similar media characteristics. If the media characteristics are dissimilar, specify a value which represents an average for OVERHEAD and DEVICE READ RATE.

Examples

- *Example 1*: Create a storage group named HIGHEND with two paths under the `/db2` directory (`/db2/filesystem1` and `/db2/filesystem2`) which are attached to Solid State Disks.

  ```
  CREATE STOGROUP HIGHEND ON '/dbe/filesystem1', '/db2/filesystem2'
     OVERHEAD 0.75 DEVICE READ RATE 500
  ```

- *Example 2*: Create a storage group named MIDRANGE with two drives D and E and designate it as the default storage group.

  ```
  CREATE STOGROUP MIDRANGE ON 'D:\', 'E:\' SET AS DEFAULT
  ```

CREATE SYNONYM

The CREATE SYNONYM statement defines a synonym for a module, nickname, sequence, table, view, or another synonym.

Description

SYNONYM is a synonym for ALIAS.

CREATE TABLE

The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns. The definition can include other attributes of the table, such as its primary key or check constraints.

To create a created temporary table, use the CREATE GLOBAL TEMPORARY TABLE statement. To declare a declared temporary table, use the DECLARE GLOBAL TEMPORARY TABLE statement.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include either DBADM authority, or CREATETAB authority in combination with further authorization, as described here:

- One of the following privileges and authorities:
  - USE privilege on the table space
  - SYSADM
  - SYSCTRL
- Plus one of these privileges and authorities:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema

If a subtable is being defined, the authorization ID must be the same as the owner of the root table of the table hierarchy.

To define a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:

- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- DBADM authority

To define a materialized query table (using a fullselect), the privileges held by the authorization ID of the statement must include at least one of the following on each table or view identified in the fullselect (excluding group privileges):

- SELECT privilege on the table or view
- CONTROL privilege on the table or view
- DATAACCESS authority

When you are defining a materialized query table and you specify certain clauses of the CREATE TABLE statement, additional authorization might be required or can be used instead:

- If WITH NO DATA is specified, at least one of the following authorities is also sufficient:
  - DBADM
  - SQLADM
  - EXPLAIN
- If REFRESH DEFERRED or REFRESH IMMEDIATE is specified, at least one of the following privileges or authority is required on each table or view identified in the fullselect:
  - ALTER privilege on the table or view
  - CONTROL privilege on the table or view
  - DBADM authority

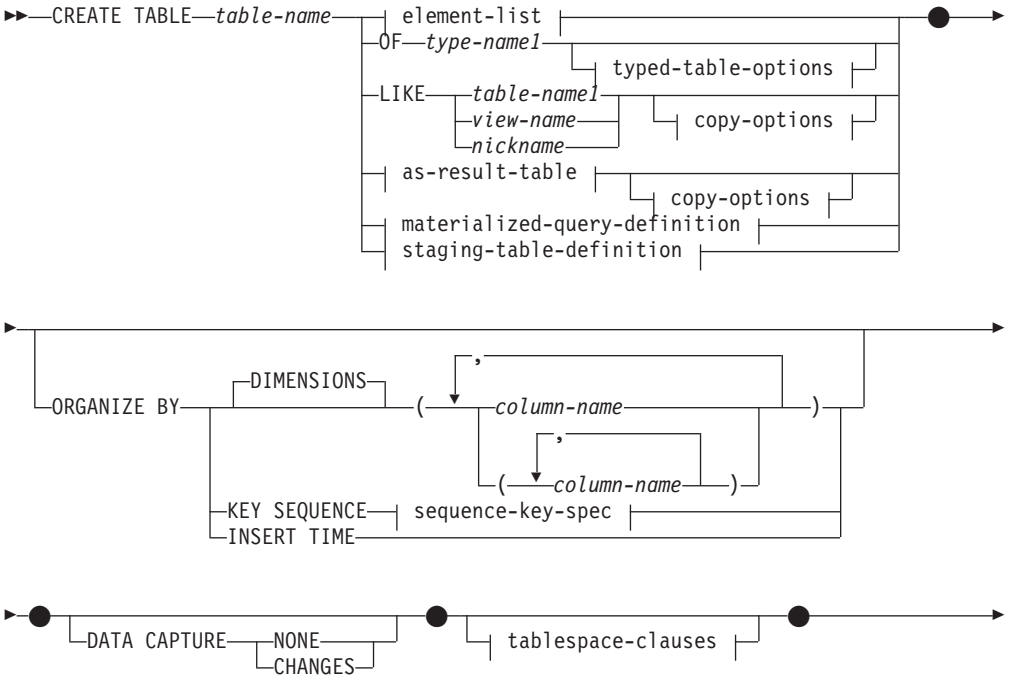
To define a staging table associated with a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following on the materialized query table:

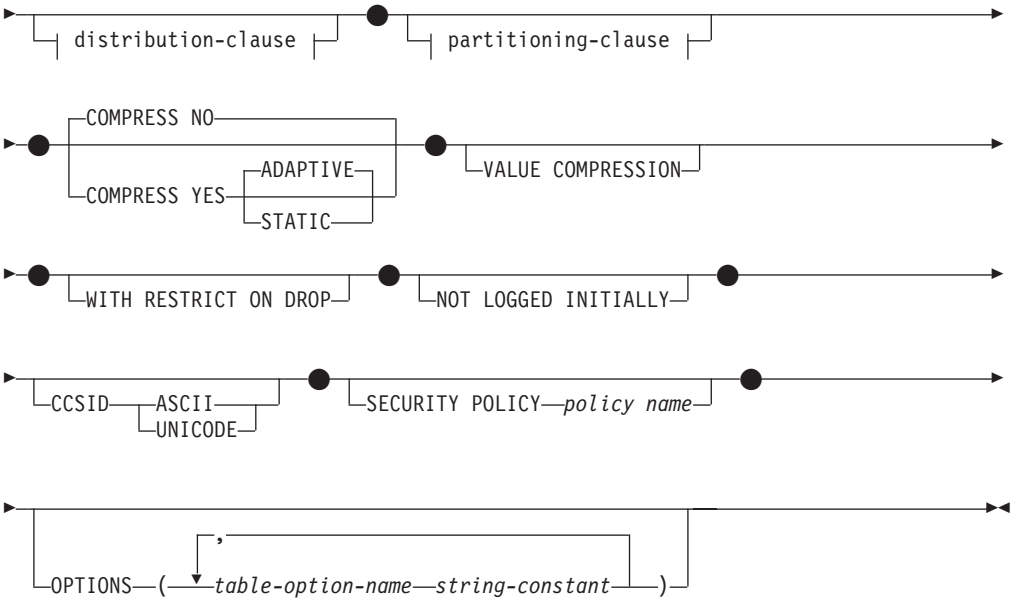
- ALTER privilege on the materialized query table
- CONTROL privilege on the materialized query table
- DBADM authority

and at least one of the following on each table or view identified in the fullselect of the materialized query table:

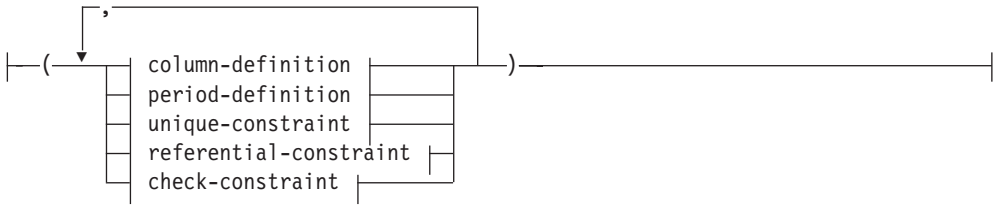
- SELECT privilege or DATAACCESS authority on the table or view, and at least one of the following:
  - ALTER privilege on the table or view
  - DBADM authority
- CONTROL privilege on the table or view

Syntax

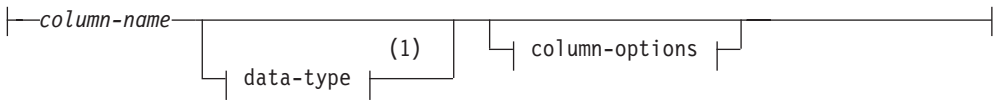




element-list:

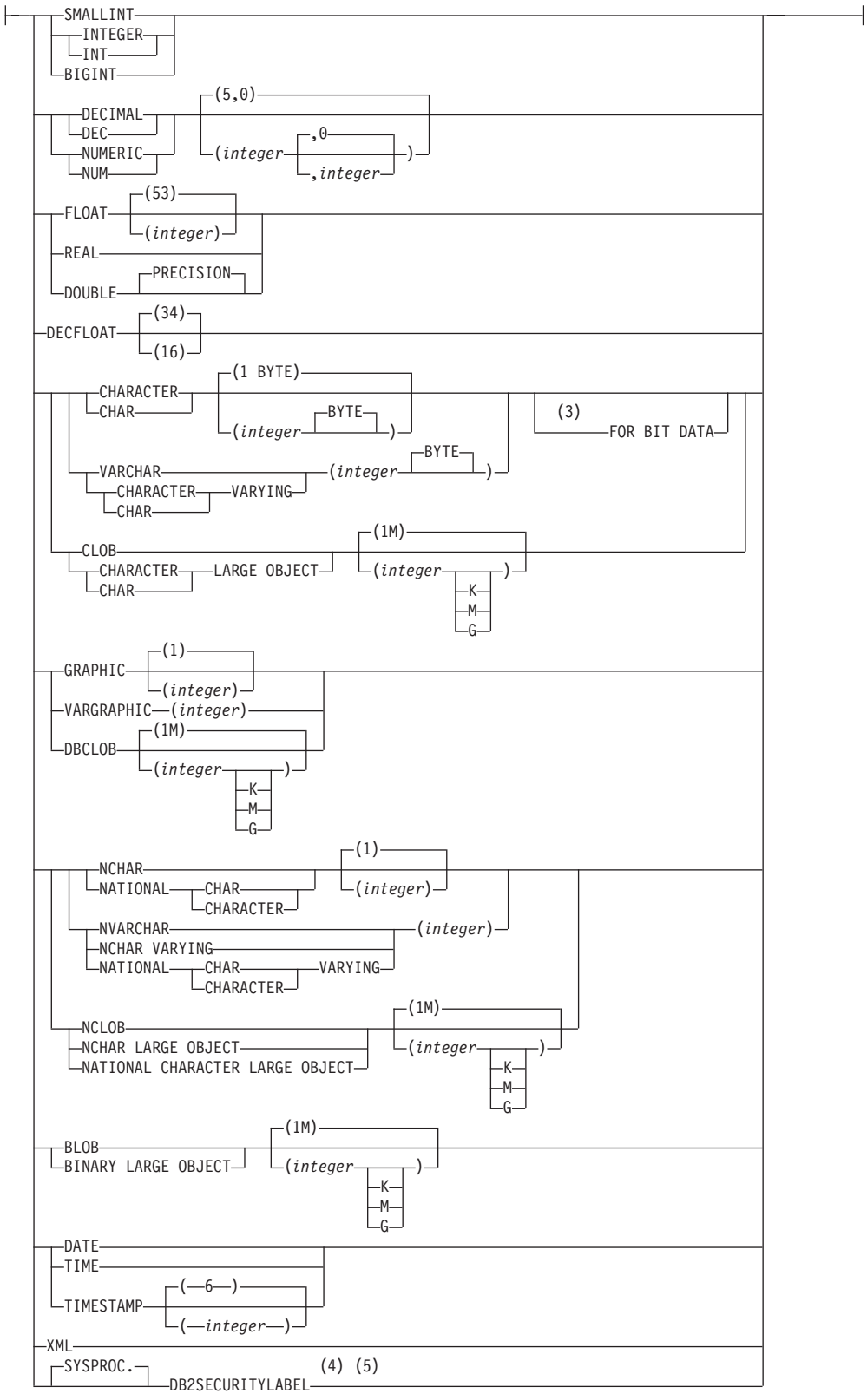


column-definition:

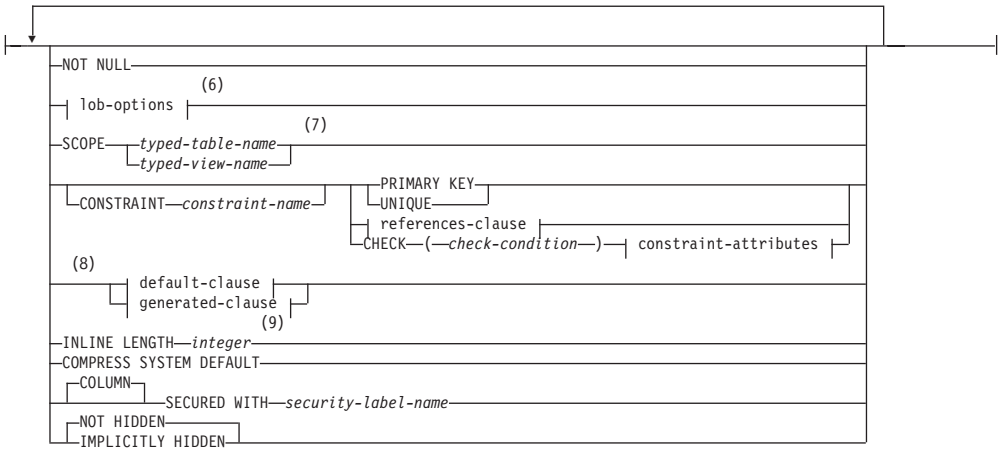


data-type:

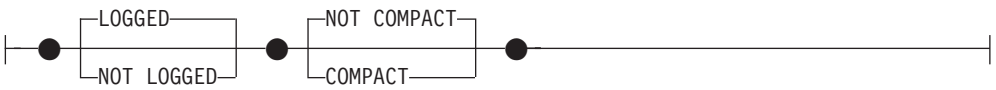
built-in-type:



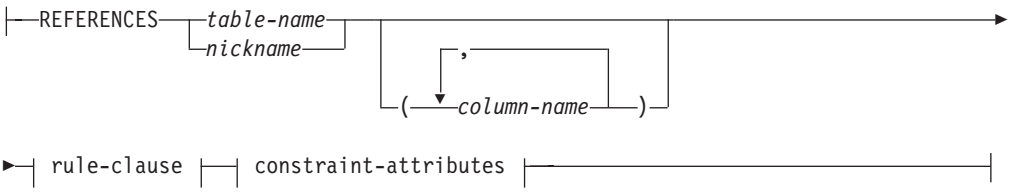
column-options:



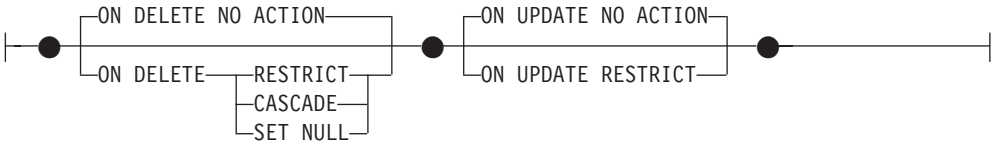
lob-options:



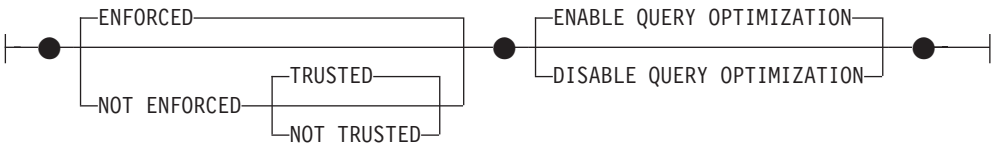
references-clause:



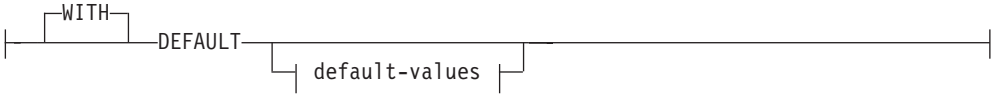
rule-clause:



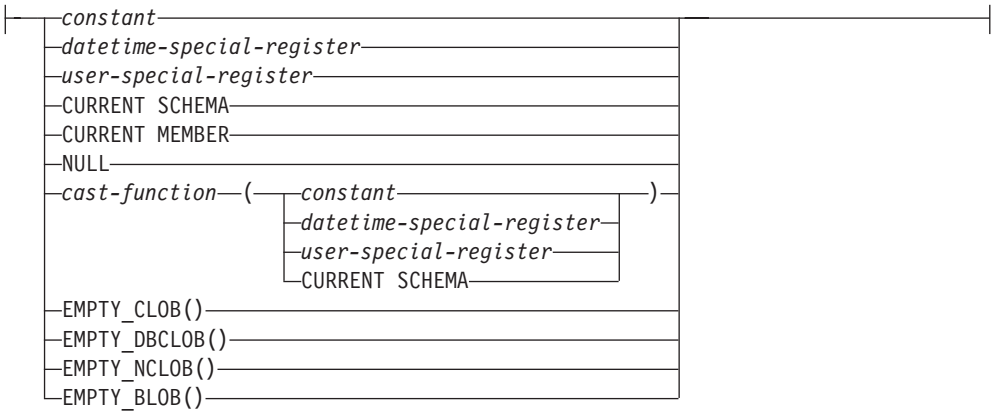
constraint-attributes:



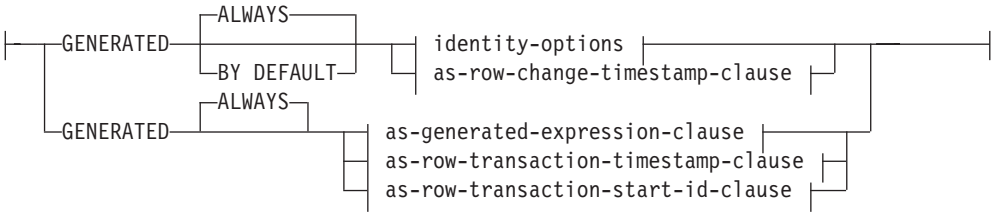
default-clause:



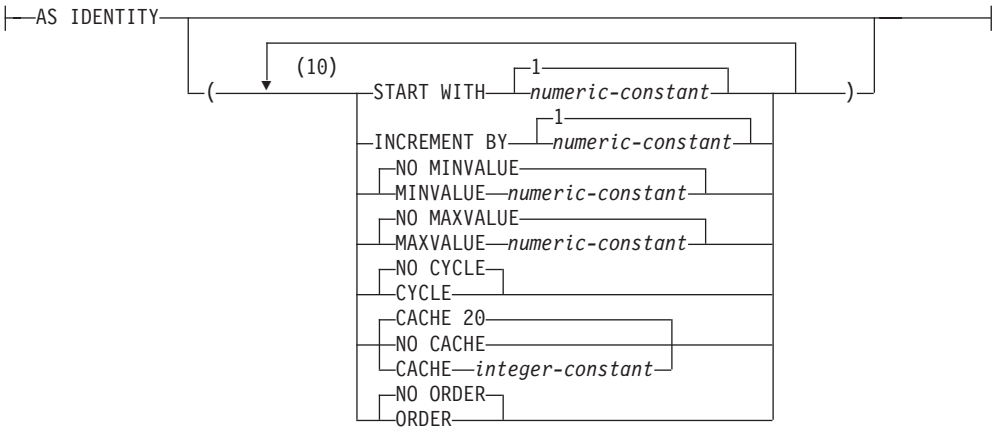
default-values:



generated-clause:



identity-options:



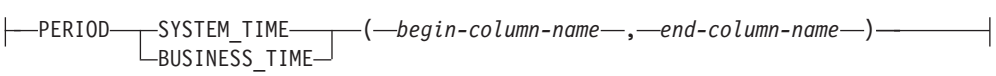
as-row-change-timestamp-clause:

as-generated-expression-clause:

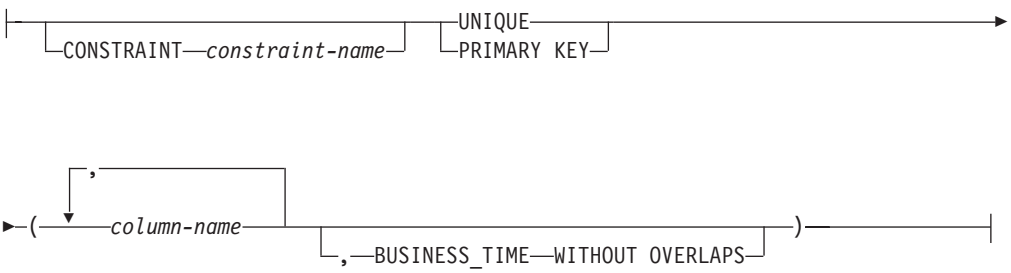
as-row-transaction-timestamp-clause:

as-row-transaction-start-id-clause:

period-definition:

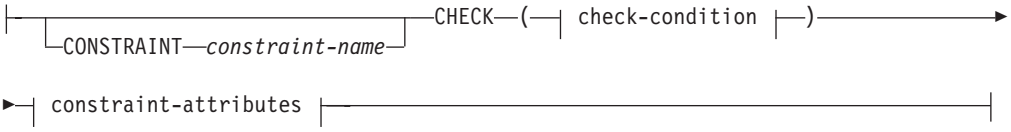


unique-constraint:



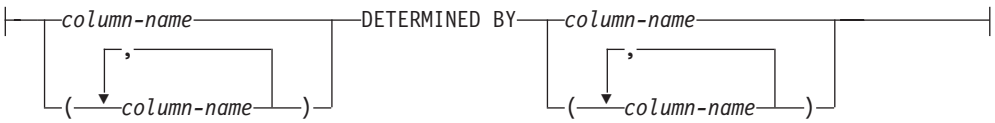
referential-constraint:

check-constraint:

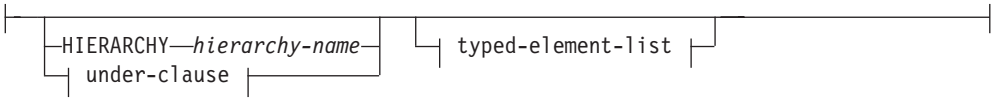


check-condition:

functional-dependency:

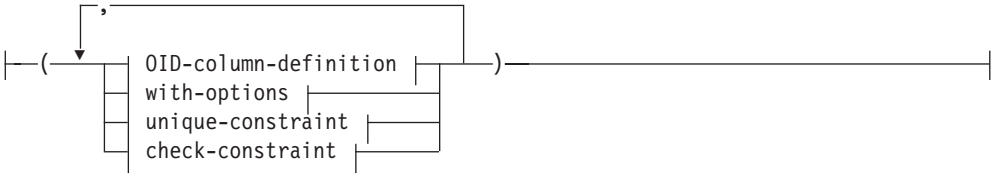


typed-table-options:



under-clause:

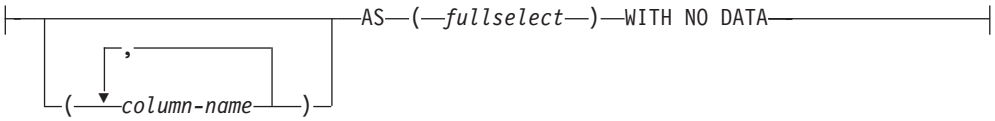
typed-element-list:



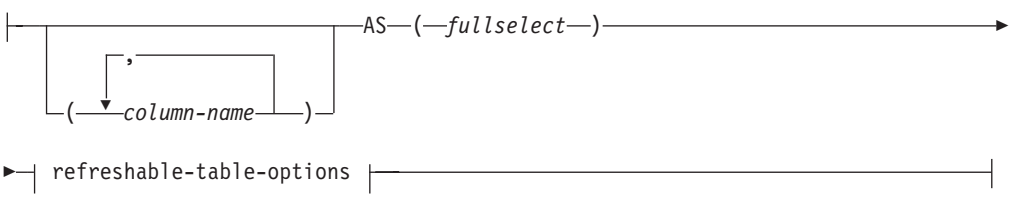
OID-column-definition:

with-options:

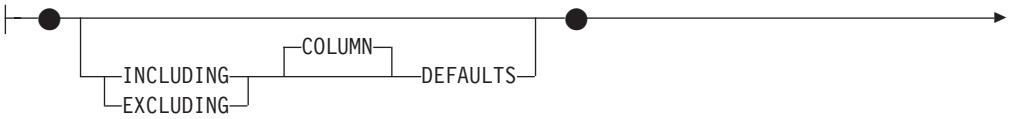
as-result-table:

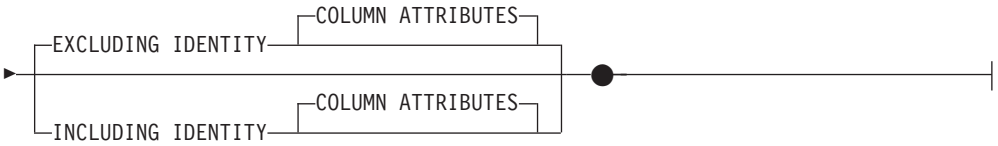


materialized-query-definition:



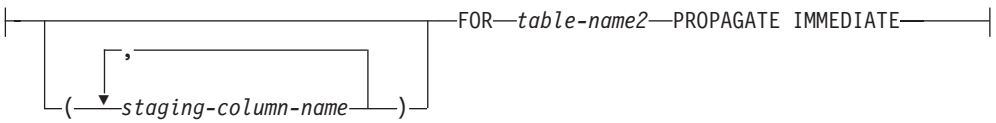
copy-options:



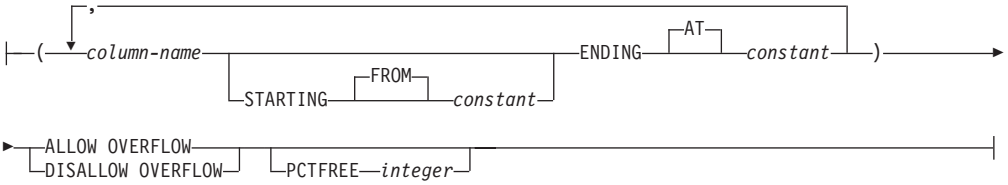


refreshable-table-options:

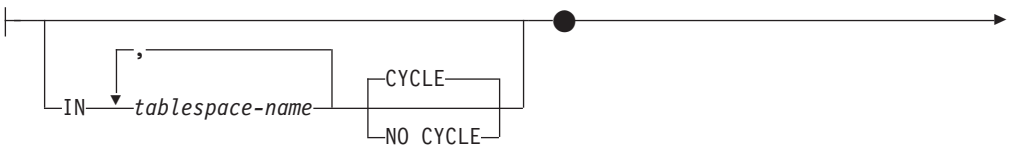
staging-table-definition:

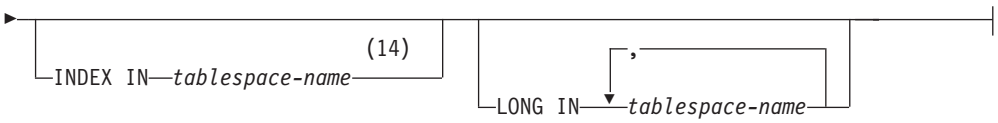


sequence-key-spec:

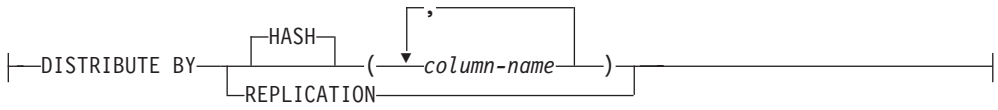


tablespace-clauses:



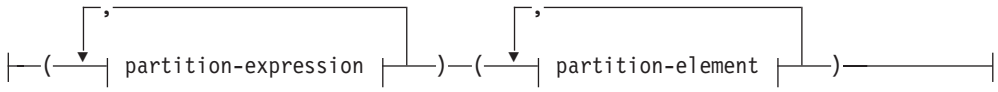


distribution-clause:



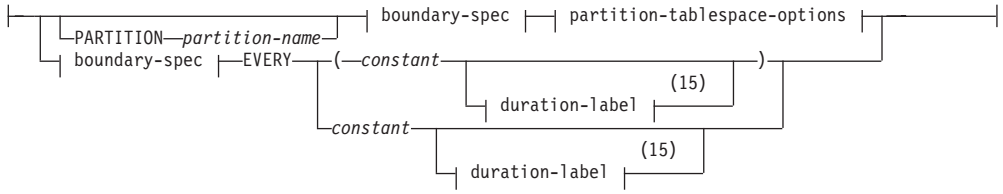
partitioning-clause:

range-partition-spec:



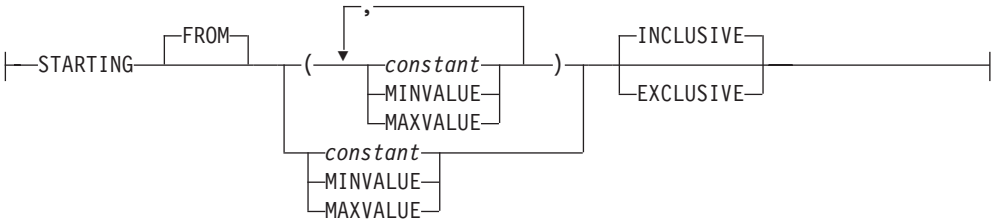
partition-expression:

partition-element:

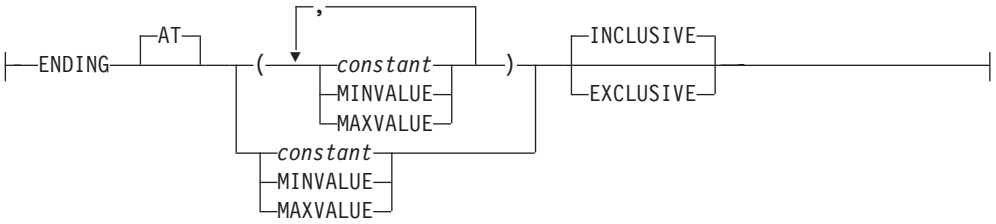


boundary-spec:

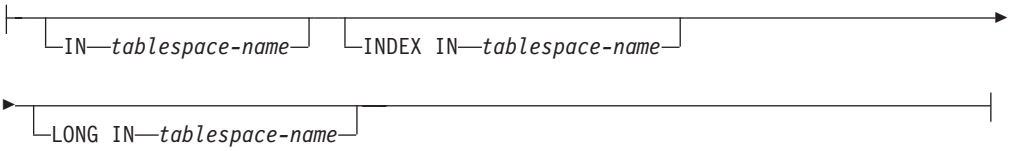
starting-clause:



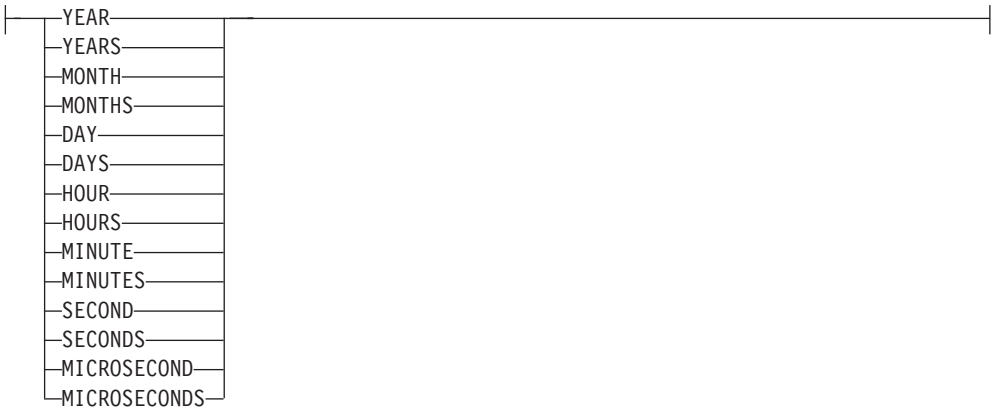
ending-clause:



partition-tablespace-options:



duration-label:



Notes:

1 If the first column-option chosen is a generated-clause with a generation-expression, then the data-type can be omitted. It will be determined from the resulting data type of the generation-expression.

2 The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type.

3 The FOR BIT DATA clause can be specified in any order with the other column constraints that follow.

4 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

5 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.

6 The lob-options clause only applies to large object types (BLOB, CLOB and DBCLOB) and distinct types based on large object types.

7 The SCOPE clause only applies to the REF type.

8 The default-clause and generated-clause cannot both be specified for the same column definition (SQLSTATE 42614).

9 INLINE LENGTH applies only to columns defined as structured, XML, or LOB types.

10 The same clause must not be specified more than once.

11 Data type is optional for a row change timestamp column if the first column-option specified is a generated-clause. The data type default is TIMESTAMP(6).

12 Data type is optional for a row-begin and row-end timestamp columns if the first column-option specified is a generated-clause. The data type default is TIMESTAMP(12).

13 Data type is optional for a transaction-start-ID timestamp columns if the first column-option specified is a generated-clause. The data type default is TIMESTAMP(12).

14 Specifying which table space will contain a table's indexes can be done when the table is created. If the table is a partitioned table, the index table space for a nonpartitioned index can be specified with the IN clause of the CREATE INDEX statement.

15 This syntax for a partition-element is valid if there is only one partition-expression with a numeric or datetime data type.

16 The first partition-element must include a starting-clause and the last partition-element must include an ending-clause.

Description

System-maintained materialized query tables and user-maintained materialized query tables are referred to by the common term *materialized query table*, unless there is a need to identify each one separately.

table-name
Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname, or alias described in the catalog. The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

element-list
Defines the elements of a table. This includes the definition of columns and constraints on the table.

column-definition
Defines the attributes of a column.

column-name
Names a column of the table. The name cannot be qualified, and the same name cannot be used for more than one column of the table (SQLSTATE 42711).

A table may have the following:

- A 4K page size with a maximum of 500 columns, where the byte counts of the columns must not be greater than 4 005.
- An 8K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 8 101.
- A 16K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 16 293.
- A 32K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 32 677.

For more details, see Row Size Limit.

data-type
Specifies the data type of the column.

built-in-type
For built-in types, use one of the following types.

SMALLINT
For a small integer.

INTEGER or INT
For a large integer.

BIGINT
For a big integer.

DECIMAL*(precision-integer, scale-integer)* **or DEC***(precision-integer, scale-integer)*
For a decimal number. The first integer is the precision of the number; that is, the total number of digits; it may range from 1 to 31. The second integer is the scale of the number; that is, the number of digits to the right of the decimal point; it may range from 0 to the precision of the number.

If precision and scale are not specified, the default values of 5,0 are used. The words **NUMERIC** and **NUM** can be used as synonyms for **DECIMAL** and **DEC**.

FLOAT(*integer***)**
For a single or double-precision floating-point number, depending on the value of the *integer*. The value of the integer must be in the range 1 through 53. The values 1 through 24 indicate single precision and the values 25 through 53 indicate double-precision.

You can also specify:

REAL For single precision floating-point.

DOUBLE
For double-precision floating-point.

DOUBLE PRECISION
For double-precision floating-point.

FLOAT
For double-precision floating-point.

DECFLOAT*(precision-integer)*
For a decimal floating-point number. The value of *precision-integer* is the precision of the number; that is, the total number of digits, which can be 16 or 34.

If the precision is not specified, a default value of 34 is used.

CHARACTER*(integer)* **or CHAR***(integer)* **or CHARACTER or CHAR**
For a fixed-length character string of length *integer* bytes, which may range from 1 to 254. If the length specification is omitted, a length of 1 is assumed.

VARCHAR*(integer)***, or CHARACTER VARYING***(integer)***, or CHAR VARYING***(integer)*
For a varying-length character string of maximum length *integer* bytes, which may range from 1 to 32 672.

FOR BIT DATA
Specifies that the contents of the column are to be treated as bit (binary) data. During data exchange with other systems, code page conversions are not performed. Comparisons are done in binary, irrespective of the database collating sequence.

CLOB or CHARACTER (CHAR) LARGE OBJECT*(integer [K | M | G])*
For a character large object string of the specified maximum length in bytes.

The meaning of the *integer K | M | G* is the same as for BLOB.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

It is not possible to specify the FOR BIT DATA clause for CLOB columns. However, a CHAR FOR BIT DATA string can be assigned to a CLOB column, and a CHAR FOR BIT DATA string can be concatenated with a CLOB string.

GRAPHIC*(integer)*
For a fixed-length graphic string of length *integer* which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed.

VARGRAPHIC*(integer)*
For a varying-length graphic string of maximum length *integer*, which may range from 1 to 16 336.

DBCLOB*(integer [K | M | G])*
For a double-byte character large object string of the specified maximum length in double-byte characters.

The meaning of the *integer K | M | G* is similar to that for BLOB. The differences are that the number specified is the number of double-byte characters, and that the maximum size is 1 073 741 823 double-byte characters.

If the length specification is omitted, a length of 1 048 576 double-byte characters is assumed.

NATIONAL CHARACTER *(integer)* **or NATIONAL CHAR** *(integer)* **or NCHAR** *(integer)*
For a fixed-length graphic string of length *integer* which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed.

NATIONAL CHARACTER VARYING *(integer)* **or NATIONAL CHAR VARYING** *(integer)* **or NCHAR VARYING** *(integer)* **or NVARCHAR** *(integer)*
For a varying-length graphic string of maximum length *integer*, which may range from 1 to 16 336.

NATIONAL CHARACTER LARGE OBJECT *(integer[K|M|G])* **or NCHAR LARGE OBJECT** *(integer[K|M|G])* **or NCLOB***(integer[K|M|G])*
For a double-byte character large object string of the specified maximum length in double-byte characters.

The meaning of the *integer* K | M | G is similar to that for BLOB. The differences are that the number specified is the number of double-byte characters, and that the maximum size is 1 073 741 823 double-byte characters.

If the length specification is omitted, a length of 1 048 576 double-byte characters is assumed.

BLOB or BINARY LARGE OBJECT*(integer [K | M | G])*
For a binary large object string of the specified maximum length in bytes.

The length may be in the range of 1 byte to 2 147 483 647 bytes.

If *integer* by itself is specified, that is the maximum length.

If *integer K* (in either upper- or lowercase) is specified, the maximum length is 1 024 times *integer*. The maximum value for *integer* is 2 097 152.

If *integer M* is specified, the maximum length is 1 048 576 times *integer*. The maximum value for *integer* is 2 048.

If *integer G* is specified, the maximum length is 1 073 741 824 times *integer*. The maximum value for *integer* is 2.

If a multiple of K, M or G that calculates out to 2 147 483 648 is specified, the actual value used is 2 147 483 647 (or 2 gigabytes minus 1 byte), which is the maximum length for a LOB column.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

To create BLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

Any number of spaces is allowed between the integer and K, M, or G, and a space is not required. For example, all of the following are valid:

```
BLOB(50K)    BLOB(50 K)    BLOB (50   K)
```

DATE
For a date.

TIME
For a time.

TIMESTAMP(*integer***) or TIMESTAMP**
For a timestamp. The *integer* must be between 0 and 12 and specifies the precision of fractional seconds from 0 (seconds) to 12 (picoseconds). The default is 6 (microseconds).

XML
For an XML document. Only well-formed XML documents can be inserted into an XML column.

An XML column has the following restrictions:

- The column cannot be part of any index except an index over XML data. Therefore, it cannot be included as a column of a primary key or unique constraint (SQLSTATE 42962).
- The column cannot be a foreign key of a referential constraint (SQLSTATE 42962).
- A default value (WITH DEFAULT) cannot be specified for the column (SQLSTATE 42613). If the column is nullable, the default for the column is the null value.
- The column cannot be used as the distribution key (SQLSTATE 42997).
- The column cannot be used as a data partitioning key (SQLSTATE 42962).
- The column cannot be used to organize a multidimensional clustering (MDC) table (SQLSTATE 42962).
- The column cannot be used in a range-clustered table (SQLSTATE 429BG).
- The column cannot be referenced in a check constraint except in a VALIDATED predicate (SQLSTATE 42621).

When a column of type XML is created, an XML path index is created on that column. A table-level XML region index is also created when the first column of type XML is created. The name of these indexes is 'SQL' followed by a character timestamp (*yymmddhhmmssxxx*). The schema name is SYSIBM.

SYSPROC.DB2SECURITYLABEL
This is a built-in distinct type that must be used to define the row security label column of a protected table. The underlying data type of a column of the built-in distinct type DB2SECURITYLABEL is VARCHAR(128) FOR BIT DATA. A table can have at most one column of type DB2SECURITYLABEL (SQLSTATE 428C1).

distinct-type-name
For a user-defined type that is a distinct type. If a distinct type name is specified without a schema name, the distinct type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a distinct type, then the data type of the column is the distinct type. The length and the scale of the column are respectively the length and the scale of the source type of the distinct type. The specified distinct type cannot have any data type constraints and the source type cannot be an anchored data type (SQLSTATE 428H2).

If a column defined using a distinct type is a foreign key of a referential constraint, then the data type of the corresponding column of the primary key must have the same distinct type.

structured-type-name
For a user-defined type that is a structured type. If a structured type

name is specified without a schema name, the structured type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL, and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a structured type, then the static data type of the column is the structured type. The column may include values with a dynamic type that is a subtype of *structured-type-name*.

A column defined using a structured type cannot be used in a primary key, unique constraint, foreign key, index key or distribution key (SQLSTATE 42962).

If a column is defined using a structured type, and contains a reference-type attribute at any level of nesting, that reference-type attribute is unscoped. To use such an attribute in a dereference operation, it is necessary to specify a SCOPE explicitly, using a CAST specification.

REF (*type-name2***)**
For a reference to a typed table. If *type-name2* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The underlying data type of the column is based on the representation data type specified in the REF USING clause of the CREATE TYPE statement for *type-name2* or the root type of the data type hierarchy that includes *type-name2*.

column-options
Defines additional options related to columns of the table.

NOT NULL
Prevents the column from containing null values.

If NOT NULL is not specified, the column can contain null values, and its default value is either the null value or the value provided by the WITH DEFAULT clause.

NOT HIDDEN or IMPLICITLY HIDDEN
Specifies whether the column is to be defined as hidden. The hidden attribute determines whether the column is included in an implicit reference to the table, or whether it can be explicitly referenced in SQL statements. The default is NOT HIDDEN.

NOT HIDDEN
Specifies that the column is included in implicit references to the table, and that the column can be explicitly referenced.

IMPLICITLY HIDDEN
Specifies that the column is not visible in SQL statements unless the column is explicitly referenced by name. For example, assuming that a table includes a column defined with the IMPLICITLY HIDDEN clause, the result of a SELECT * does not include the implicitly hidden column. However, the result of a SELECT that explicitly refers to the name of an implicitly hidden column will include that column in the result table.

IMPLICITLY HIDDEN must not be specified for all columns of the table (SQLSTATE 428GU).

lob-options
Specifies options for LOB data types.

LOGGED
Specifies that changes made to the column are to be written to the log. The data in such columns is then recoverable with database utilities (such as RESTORE DATABASE). LOGGED is the default.

NOT LOGGED
Specifies that changes made to the column are not to be logged. This only applies to LOB data that is not inlined.

NOT LOGGED has no effect on a commit or rollback operation; that is, the database's consistency is maintained even if a transaction is rolled back, regardless of whether or not the LOB value is logged. The implication of not logging is that during a roll forward operation, after a backup or load operation, the LOB data will be replaced by zeros for those LOB values that would have had log records replayed during the roll forward. During crash recovery, all committed changes and changes rolled back will reflect the expected results.

COMPACT
Specifies that the values in the LOB column should take up minimal disk space (free any extra disk pages in the last group used by the LOB value), rather than leave any leftover space at the end of the LOB storage area that might facilitate subsequent append operations. Note that storing data in this way may cause a performance penalty in any append (length-increasing) operations on the column.

NOT COMPACT
Specifies some space for insertions to assist in future changes to the LOB values in the column. This is the default.

SCOPE
Identifies the scope of the reference type column.

A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the DEREF function. Specifying the scope for a reference type column may be deferred to a subsequent ALTER TABLE statement to allow the target table to be defined, usually in the case of mutually referencing tables.

typed-table-name
The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same CREATE TABLE statement. (SQLSTATE 42710).

If this clause is omitted, an 18 byte long identifier that is unique among the identifiers of existing constraints defined on the table is generated by the system. (The identifier consists of "SQL" followed by a sequence of 15 numeric characters generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) because the primary key is inherited from the supertable.

A ROW CHANGE TIMESTAMP column cannot be used as part of a primary key (SQLSTATE 429BV).

Row-begin, row-end, and transaction-start-ID columns cannot be used as part of a primary key (SQLSTATE 429BV).

See PRIMARY KEY within the *unique-constraint* description.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3) since unique constraints are inherited from the supertable.

See UNIQUE within the *unique-constraint* description.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* under *referential-constraint* description.

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See description for CHECK (*check-condition*).

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT

Provides a default value in the event a value is not supplied on insert or is specified as DEFAULT on INSERT or UPDATE. If a default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in "ALTER TABLE". This clause must not be specified with generated-clause in a column definition (SQLSTATE 42614).

If a column is defined as XML, a default value cannot be specified (SQLSTATE 42613). The only possible default is NULL.

If the column is based on a column of a typed table, a specific default value must be specified when defining a default. A default value cannot be specified for the object identifier column of a typed table (SQLSTATE 42997).

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

If a column is defined using a structured type, the *default-clause* cannot be specified (SQLSTATE 42842).

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

default-values
Specific types of default values that can be specified are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment
- not be a floating-point constant unless the column is defined with a floating-point data type
- be a numeric constant or a decimal floating-point special value if the data type of the column is a decimal floating-point. Floating-point constants are first interpreted as DOUBLE and then converted to decimal floating-point if the target column is DECFLOAT. For DECFLOAT(16) columns, decimal constants having precision greater than 16 digits will be rounded using the rounding modes specified by the CURRENT DECFLOAT ROUNDING MODE special register.
- not have nonzero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the

column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register.

CURRENT MEMBER
Specifies the value of the CURRENT MEMBER special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT MEMBER is specified, the data type of the column must allow assignment from an integer.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL may be specified within the same column definition but will result in an error on any attempt to set the column to the default value.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

EMPTY_CLOB(), EMPTY_DBCLOB(), or EMPTY_BLOB()
Specifies a zero-length string as the default for the column. The column must have the data type that corresponds to the result data type of the function.

If the value specified is not valid, an error is returned (SQLSTATE 42894).

generated-clause
Specifies a generated value for the column.

GENERATED

Specifies that DB2 generates values for the column. GENERATED must be specified if the column is to be considered an identity column or a row change timestamp column, row-begin column, row-end column, transaction-start-ID column, or generated expression column. A default clause must not be specified for a column defined as GENERATED (SQLSTATE 42623).

ALWAYS
Specifies that a value will always be generated for the column when a row is inserted into the table, or whenever the result value of the *generation-expression* changes. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended value unless data propagation or unload and reload operations are being done. GENERATED ALWAYS is the required value for generated columns.

BY DEFAULT
Specifies that DB2 will generate a value for the column when a row is inserted, or updated specifying the DEFAULT clause, unless an explicit value is specified. BY DEFAULT is the recommended value when using data propagation or performing an unload and reload operation.

Although not explicitly required, to ensure uniqueness of the values, define a unique single-column index on generated IDENTITY columns.

AS IDENTITY
Specifies that the column is to be the identity column for this table. A table can only have a single identity column (SQLSTATE 428C1). The IDENTITY keyword can only be specified if the data type associated with the column is an exact numeric type with a scale of zero, or a user-defined distinct type for which the source type is an exact numeric type with a scale of zero (SQLSTATE 42815). SMALLINT, INTEGER, BIGINT, or DECIMAL with a scale of zero, or a distinct type based on one of these types, are considered exact numeric types. By contrast, single- and double-precision floating points are considered approximate numeric data types. Reference types, even if represented by an exact numeric type, cannot be defined as identity columns.

An identity column is implicitly NOT NULL. An identity column cannot have a DEFAULT clause (SQLSTATE 42623).

START WITH *numeric-constant*
Specifies the first value for the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA). The default is MINVALUE for ascending sequences, and MAXVALUE for descending sequences. This value is not necessarily the value that would be cycled to after reaching the maximum or minimum value for the identity column. The START WITH clause can be used to start the generation of values outside the range that is used for cycles. The range used for cycles is defined by MINVALUE and MAXVALUE.

INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence. If this value is 0, or positive, this is an ascending sequence. The default is 1.

NO MINVALUE or MINVALUE
Specifies the minimum value at which a descending identity column either cycles or stops generating values, or an ascending identity column cycles to after reaching the maximum value.

NO MINVALUE
For an ascending sequence, the value is the START WITH value, or 1 if START WITH was not specified. For a descending sequence, the value is the minimum value of the data type of the column. This is the default.

MINVALUE *numeric-constant*
Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point

(SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

NO MAXVALUE or MAXVALUE
Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or a descending identity column cycles to after reaching the minimum value.

NO MAXVALUE
For an ascending sequence, the value is the maximum value of the data type of the column. For a descending sequence, the value is the START WITH value, or -1 if START WITH was not specified. This is the default.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without nonzero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

NO CYCLE or CYCLE
Specifies whether this identity column should continue to generate values after generating either its maximum or minimum value.

NO CYCLE
Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached. This is the default.

CYCLE
Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, DB2 may generate duplicate values for an identity column. Although not explicitly required, a unique, single-column index should be defined on the generated column to ensure uniqueness of the values, if unique values are required. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

NO CACHE or CACHE
Specifies whether to keep some pre-allocated values in memory for faster access. If a new value is needed for the identity column, and there are none available in the cache, then the end of the new cache block must be logged. However, when a new value is needed for the identity column, and there is an unused value in the cache, then the allocation of that identity value is faster, because no logging is necessary. This is a performance and tuning option.

NO CACHE
Specifies that values for the identity column are not to be pre-allocated.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*
Specifies how many values of the identity sequence are to be pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value involves waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used in committed statements are *lost*; that is, they will never be used. The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of database deactivation. (If a database is not explicitly activated, using the ACTIVATE command or API, when the last application is disconnected from the database, an implicit deactivation occurs.)

The minimum value is 2 (SQLSTATE 42815). The default value is CACHE 20.

In a multi-partition or DB2 pureScale environment, use the CACHE and NO ORDER options to allow multiple DB2 members to cache sequence values simultaneously.

In a DB2 pureScale environment, if both CACHE and ORDER are specified, the specification of ORDER overrides the specification of CACHE and instead NO CACHE will be in effect.

NO ORDER or ORDER
Specifies whether the identity values must be generated in order of request.

NO ORDER
Specifies that the values do not need to be generated in order of request. This is the default.

ORDER
Specifies that the values must be generated in order of request.

FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
Specifies that the column is a timestamp column for the table. A

value is generated for the column in each row that is inserted, and for any row in which any column is updated. The value that is generated for a ROW CHANGE TIMESTAMP column is a timestamp that corresponds to the insert or update time for that row. If multiple rows are inserted or updated with a single statement, the value of the ROW CHANGE TIMESTAMP column might be different for each row.

A table can only have one ROW CHANGE TIMESTAMP column (SQLSTATE 428C1). If *data-type* is specified, it must be TIMESTAMP or TIMESTAMP(6) (SQLSTATE 42842). A ROW CHANGE TIMESTAMP column cannot have a DEFAULT clause (SQLSTATE 42623). NOT NULL must be specified for a ROW CHANGE TIMESTAMP column (SQLSTATE 42831).

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. (If the expression for a GENERATED ALWAYS column includes a user-defined external function, changing the executable for the function (such that the results change for given arguments) can result in inconsistent data. This can be avoided by using the SET INTEGRITY statement to force the generation of new values.) The *generation-expression* cannot contain any of the following (SQLSTATE 42621):

- Subqueries
- XMLQUERY or XMLEXISTS expressions
- Column functions
- Dereference operations or DEREF functions
- User-defined or built-in functions that are non-deterministic
- User-defined functions using the EXTERNAL ACTION option
- User-defined functions that are not defined with NO SQL
- Host variables or parameter markers
- Special registers and built-in functions that depend on the value of a special register
- Global variables
- References to columns defined later in the column list
- References to other generated columns
- References to columns of type XML

The data type for the column is based on the result data type of the *generation-expression*. A CAST specification can be used to force a particular data type and to provide a scope (for a reference type only). If *data-type* is specified, values are assigned to the column according to the appropriate assignment rules. A generated column is implicitly considered nullable, unless the NOT NULL column option is used. The data type of a generated column and the result data type of the *generation-expression* must have equality defined (see "Assignments and comparisons"). This excludes columns and generation expressions of type LOB data types, XML, structured types, and distinct types based on any of these types (SQLSTATE 42962).

AS ROW BEGIN

Specifies that the generated value is assigned by the database manager whenever a row is inserted into the table or any column in the row is updated. The value is generated using a reading from the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to the row-begin column or transaction-start-ID column in the table, or a row in a system-period temporal table is deleted.

For a system-period temporal table, the database manager ensures uniqueness of the generated values for a row-begin column across transactions. The timestamp value might be adjusted to ensure that rows inserted into an associated history table have the end timestamp value greater than the begin timestamp value. This can happen when a conflicting transaction is updating the same row in the system-period temporal table. The database configuration parameter **systime_period_adj** must be set to `Yes` for this adjustment to the timestamp value to occur. If multiple rows are inserted or updated within a single SQL transaction and an adjustment is not needed, the values for the row-begin column are the same for all the rows and are unique from the values generated for the column for another transaction. A row-begin column is required as the begin column of a SYSTEM_TIME period, which is the intended use for this type of generated column.

A table can have only one row-begin column (SQLSTATE 428C1). If *data-type* is not specified the column is defined as a TIMESTAMP(12). If *data-type* is specified, it must be TIMESTAMP(12) (SQLSTATE 42842). The column cannot have a DEFAULT clause (SQLSTATE 42623), and must be defined as NOT NULL (SQLSTATE 42831). A row-begin column is not updatable.

AS ROW END

Specifies that the maximum value for the data type of the column (9999-12-30-00.00.00.000000000000) is assigned by the database manager whenever a row is inserted or any column in the row is updated.

A row-end column is required as the second column of a SYSTEM_TIME period, which is the intended use for this type of generated column.

A table can have only one row-end column (SQLSTATE 428C1). If *data-type* is not specified, the column is defined as TIMESTAMP(12). If *data-type* is specified, it must be TIMESTAMP(12) (SQLSTATE 42842). The column cannot have a DEFAULT clause (SQLSTATE 42623), and must be defined as NOT NULL (SQLSTATE 42831). A row-end column is not updatable.

AS TRANSACTION START ID

Specifies that the value is assigned by the database manager whenever a row is inserted into the table or any column in the row is updated. The database manager assigns a unique timestamp value per transaction or the null value. The null value is assigned to the transaction-start-ID column if the column is nullable and if there is a row-begin column in the table for which the value did not need to be adjusted. Otherwise the value is generated using a reading of the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be

assigned to a row-begin column or transaction-start-ID column in the table, or a row in a system-period temporal table is deleted. If multiple rows are inserted or updated within a single SQL transaction, the values for the transaction-start-ID column are the same for all the rows and are unique from the values generated for the column for another transaction.

A transaction-start-ID column is required for a system-period temporal table, which is the intended use for this type of generated column.

A table can have only one transaction-start-ID column (SQLSTATE 428C1). If *data-type* is not specified, the column is defined as TIMESTAMP(12). If *data-type* is specified it must be TIMESTAMP(12). A transaction-start-ID column cannot have a DEFAULT clause (SQLSTATE 42623). A transaction-start-ID column is not updatable.

INLINE LENGTH *integer*

This option is valid only for a column defined using a structured type, XML or LOB data type (SQLSTATE 42842).

For a column of data type XML or LOB, *integer* indicates the maximum byte size of the internal representation of an XML document or LOB data to store in the base table row. XML documents that have a larger internal representation are stored separately from the base table row in an auxiliary storage object. This takes place automatically. There is no default inline length for XML type columns. If the XML document or LOB data is stored inlined in the base table row, there is an additional overhead. For LOB data, the overhead is 4 bytes.

For a column of data type LOB, the default inline length is set to be the maximum size of the LOB descriptor if the clause is not specified. Any explicit INLINE LENGTH must be at least the maximum LOB descriptor size. The following table summarizes the LOB descriptor sizes.

Table 19. Sizes of the LOB descriptor for various LOB lengths

Maximum LOB length in bytes	Minimum explicit INLINE LENGTH
1,024	68
8,192	92
65,536	116
524,000	140
4,190,000	164
134,000,000	196
536,000,000	220
1,070,000,000	252
1,470,000,000	276
2,147,483,647	312

For a structured type column, *integer* indicates the maximum size in bytes of an instance of a structured type to store inline with the rest of the values in the row. Instances of structured types that cannot be stored inline are stored separately from the base table row, similar to the way that LOB values are stored. This takes place automatically. The

default INLINE LENGTH for a structured-type column is the inline length of its type (specified explicitly or by default in the CREATE TYPE statement). If INLINE LENGTH of the structured type is less than 292, the value 292 is used for the INLINE LENGTH of the column.

Note: The inline lengths of subtypes are not counted in the default inline length, meaning that instances of subtypes may not fit inline unless an explicit INLINE LENGTH is specified at CREATE TABLE time to account for existing and future subtypes.

The explicit INLINE LENGTH value cannot exceed 32 673. For a structured type or XML data type, it must be at least 292 (SQLSTATE 54010).

COMPRESS SYSTEM DEFAULT
Specifies that system default values are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned (SQLSTATE 01648), and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be a DATE, TIME, TIMESTAMP, XML, or structured data type (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

COLUMN SECURED WITH *security-label-name*
Identifies a security label that exists for the security policy that is associated with the table. The name must not be qualified (SQLSTATE 42601). The table must have a security policy associated with it (SQLSTATE 55064). The table must not be a system-period temporal table.

period-definition

PERIOD
Defines a period for the table.

SYSTEM_TIME (*begin-column-name***, *end-column-name***)**

Defines a system period with the name SYSTEM_TIME. There must not be a column in the table with the name SYSTEM_TIME (SQLSTATE 42711). A table can have only one SYSTEM_TIME period (SQLSTATE 42711). *begin-column-name* must be defined as ROW BEGIN and *end-column-name* must be defined as ROW END (SQLSTATE 428HN).

BUSINESS_TIME (*begin-column-name***, *end-column-name***)**

Defines an application period with the name BUSINESS_TIME. There must not be a column in the table with the name BUSINESS_TIME (SQLSTATE 42711). A table can have only one BUSINESS_TIME period (SQLSTATE 42711). *begin-column-name* and *end-column-name* must both be defined as DATE or TIMESTAMP(*p*) where *p* is from 0 to 12 (SQLSTATE 42842), and the columns must be defined as NOT NULL

(SQLSTATE 42831). *begin-column-name* and *end-column-name* must not identify a column that is defined with a GENERATED clause (SQLSTATE 428HZ).

An implicit check constraint is generated to ensure that the value of *end-column-name* is greater than the value of *begin-column-name*. The name of the implicitly created check constraint is DB2_GENERATED_CHECK_CONSTRAINT_FOR_BUSINESS_TIME and must not be name of any other check constraint specified in the statement (SQLSTATE 42710).

unique-constraint

Defines a unique or primary key constraint. If the table has a distribution key, any unique or primary key must be a superset of the distribution key. A unique or primary key constraint cannot be specified for a table that is a subtable (SQLSTATE 429B3). Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE). If the table is a root table, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*

Names the primary key or unique constraint.

UNIQUE (*column-name***, ...)**

Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once.

If the table has a BUSINESS_TIME period defined, BUSINESS_TIME WITHOUT OVERLAPS can be specified as the last item in the key expression list. If BUSINESS_TIME WITHOUT OVERLAPS is specified, the list must include at least one *column-name*. WITHOUT OVERLAPS means that for the other specified keys, the values are unique with respect to time for the BUSINESS_TIME period. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the columns of the BUSINESS_TIME period must not be specified as part of the constraint (SQLSTATE 428HW). The specification of BUSINESS_TIME WITHOUT OVERLAPS adds the following attributes to the constraint:

- The end column of the BUSINESS_TIME period in ascending order
- The begin column of the BUSINESS_TIME period in ascending order

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see Byte Counts. For key length limits, see “SQL limits”. No LOB, XML, distinct type based on one of these types, or structured type can be used as part of a unique key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the unique key cannot be the same as the set of columns in the primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3), because unique constraints are inherited from the supertable.

The description of the table as recorded in the catalog includes the unique key and its unique index. A unique bidirectional index, which allows forward and reverse scans, will automatically be created for the columns in

the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

PRIMARY KEY (*column-name***,...)**
Defines a primary key composed of the identified columns. The clause must not be specified more than once, and the identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table, and the same column must not be identified more than once.

If the table has a BUSINESS_TIME period defined, BUSINESS_TIME WITHOUT OVERLAPS can be specified as the last item in the key expression list. If BUSINESS_TIME WITHOUT OVERLAPS is specified, the list must include at least one *column-name*. WITHOUT OVERLAPS means that for the rest of the specified keys, the values are unique with respect to time for the BUSINESS_TIME period. When BUSINESS_TIME WITHOUT OVERLAPS is specified, the columns of the BUSINESS_TIME period must not be specified as part of the constraint (SQLSTATE 428HW). The specification of BUSINESS_TIME WITHOUT OVERLAPS adds the following attributes to the constraint:

- The end column of the BUSINESS_TIME period in ascending order
- The begin column of the BUSINESS_TIME period in ascending order

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see Byte Counts. For key length limits, see "SQL limits". No LOB, XML, distinct type based on one of these types, or structured type can be used as part of a primary key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

Only one primary key can be defined on a table.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) because the primary key is inherited from the supertable.

The description of the table as recorded in the catalog includes the primary key and its primary index. A unique bidirectional index, which allows forward and reverse scans, will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

If the table has a distribution key, the columns of a *unique-constraint* must be a superset of the distribution key columns; column order is unimportant.

referential-constraint
Defines a referential constraint.

CONSTRAINT *constraint-name*
Names the referential constraint.

FOREIGN KEY (*column-name***,...)**
Defines a referential constraint with the specified *constraint-name*.

Let T1 denote the object table of the statement. The foreign key of the referential constraint is composed of the identified columns. Each name in the list of column names must identify a column of T1 and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see Byte Counts. For key length limits, see "SQL limits". No LOB, XML, distinct type based on one of these types, or structured type column can be used as part of a foreign key (SQLSTATE 42962). There must be the same number of foreign key columns as there are in the parent key and the data types of the corresponding columns must be compatible (SQLSTATE 42830). Two column descriptions are compatible if they have compatible data types (both columns are numeric, character strings, graphic, date/time, or have the same distinct type).

references-clause
Specifies the parent table or the parent nickname, and the parent key for the referential constraint.

REFERENCES *table-name* **or** *nickname*
The table or nickname specified in a REFERENCES clause must identify a base table or a nickname that is described in the catalog, but must not identify a catalog table.

A referential constraint is a duplicate if its foreign key, parent key, and parent table or parent nickname are the same as the foreign key, parent key, and parent table or parent nickname of a previously specified referential constraint. Duplicate referential constraints are ignored, and a warning is returned (SQLSTATE 01543).

In the following discussion, let T2 denote the identified parent table, and let T1 denote the table being created (or altered). (T1 and T2 may be the same table).

The specified foreign key must have the same number of columns as the parent key of T2 and the description of the *n*th column of the foreign key must be comparable to the description of the *n*th column of that parent key. Datetime columns are not considered to be comparable to string columns for the purposes of this rule.

(*column-name***,...)**
The parent key of a referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of T2. The same column must not be identified more than once.

The list of column names must match the set of columns (in any order) of the primary key or a unique constraint that exists on T2 (SQLSTATE 42890). If a column name list is not specified, then T2 must have a primary key (SQLSTATE 42888). Omission of the column name list is an implicit specification of the columns of that primary key in the sequence originally specified.

The referential constraint specified by a FOREIGN KEY clause defines a relationship in which T2 is the parent and T1 is the dependent.

rule-clause
Specifies what action to take on dependent tables.

ON DELETE
Specifies what action is to take place on the dependent tables when a row of the parent table is deleted. There are four possible actions:

- NO ACTION (default)
- RESTRICT
- CASCADE
- SET NULL

The delete rule applies when a row of T2 is the object of a DELETE or propagated delete operation and that row has dependents in T1. Let *p* denote such a row of T2.

- If RESTRICT or NO ACTION is specified, an error occurs and no rows are deleted.
- If CASCADE is specified, the delete operation is propagated to the dependents of *p* in T1.
- If SET NULL is specified, each nullable column of the foreign key of each dependent of *p* in T1 is set to null.

SET NULL must not be specified unless some column of the foreign key allows null values. Omission of the clause is an implicit specification of ON DELETE NO ACTION.

If T1 is delete-connected to T2 through multiple paths, defining two SET NULL rules with overlapping foreign key definitions is not allowed. For example: T1 (i1, i2, i3). Rule1 with foreign key (i1, i2) and Rule2 with foreign key (i2, i3) is not allowed.

The firing order of the rules is:

1. RESTRICT
2. SET NULL OR CASCADE
3. NO ACTION

If any row in T1 is affected by two different rules, error occurs and no rows are deleted.

A referential constraint cannot be defined if it would cause a table to be delete-connected to itself by a cycle involving two or more tables, and where one of the delete rules is RESTRICT or SET NULL (SQLSTATE 42915).

A referential constraint that would cause a table to be delete-connected to either itself or another table by multiple paths can be defined, except in the following cases (SQLSTATE 42915):

- A table must not be both a dependent table in a CASCADE relationship (self-referencing, or referencing another table), and have a self-referencing relationship in which the delete rule is RESTRICT or SET NULL.
- A key overlaps another key when at least one column in one key is the same as a column in the other key. When a table is delete-connected to another table through multiple relationships with overlapping foreign keys, those relationships must have the same delete rule, and none of the delete rules can be SET NULL.
- When a table is delete-connected to another table through multiple relationships, and at least one of those relationships is

specified with a delete rule of SET NULL, the foreign key definitions of these relationships must not contain any distribution key or multidimensional clustering (MDC) key column.

- When two tables are delete-connected to the same table through CASCADE relationships, the two tables must not be delete-connected to each other if the delete rule of the last relationship in each delete-connected path is RESTRICT or SET NULL.

If any row in T1 is affected by different delete rules, the result would be the effect of all the actions specified by these rules. AFTER triggers and CHECK constraints on T1 will also see the effect of all the actions. An example of this is a row that is targeted to be set null through one delete-connected path to an ancestor table, and targeted to be deleted by a second delete-connected path to the same ancestor table. The result would be the deletion of the row. AFTER DELETE triggers on this descendant table would be activated, but AFTER UPDATE triggers would not.

In applying the previously mentioned rules to referential constraints, in which either the parent table or the dependent table is a member of a typed table hierarchy, all the referential constraints that apply to any table in the respective hierarchies are taken into consideration.

ON UPDATE
Specifies what action is to take place on the dependent tables when a row of the parent table is updated. The clause is optional. ON UPDATE NO ACTION is the default and ON UPDATE RESTRICT is the only alternative.

The difference between NO ACTION and RESTRICT is described in the "Notes" section.

check-constraint
Defines a check constraint. A *check-constraint* is a *search-condition* that must evaluate to not false or a functional dependency that is defined between columns.

CONSTRAINT *constraint-name*
Names the check constraint.

CHECK (*check-condition***)**
Defines a check constraint. The *search-condition* must be true or unknown for every row of the table.

search-condition
The *search-condition* has the following restrictions:

- A column reference must be to a column of the table being created.
- The *search-condition* cannot contain a TYPE predicate.
- The *search-condition* cannot contain any of the following (SQLSTATE 42621):
  - Subqueries
  - XMLQUERY or XMLEXISTS expressions
  - Dereference operations or DEREF functions where the scoped reference argument is other than the object identifier (OID) column

- CAST specifications with a SCOPE clause
- Column functions
- Functions that are not deterministic
- Functions defined to have an external action
- User-defined functions defined with either MODIFIES SQL or READS SQL DATA
- Host variables
- Parameter markers
- *sequence-references*
- OLAP specifications
- Special registers and built-in functions that depend on the value of a special register
- Global variables
- References to generated columns other than the identity column
- References to columns of type XML (except in a VALIDATED predicate)
- An error tolerant *nested-table-expression*

functional-dependency
Defines a functional dependency between columns.

column-name **DETERMINED BY** *column-name* **or (***column-name***,...)**
DETERMINED BY (*column-name***,...)**
The parent set of columns contains the identified columns that immediately precede the DETERMINED BY clause. The child set of columns contains the identified columns that immediately follow the DETERMINED BY clause. All of the restrictions on the *search-condition* apply to parent set and child set columns, and only simple column references are allowed in the set of columns (SQLSTATE 42621). The same column must not be identified more than once in the functional dependency (SQLSTATE 42709). The data type of the column must not be a LOB data type, a distinct type based on a LOB data type, an XML data type, or a structured type (SQLSTATE 42962). A ROW CHANGE TIMESTAMP column cannot be used as part of a primary key (SQLSTATE 429BV). No column in the child set of columns can be a nullable column (SQLSTATE 42621).

If a check constraint is specified as part of a *column-definition*, a column reference can only be made to the same column. Check constraints specified as part of a table definition can have column references identifying columns previously defined in the CREATE TABLE statement. Check constraints are not checked for inconsistencies, duplicate conditions, or equivalent conditions. Therefore, contradictory or redundant check constraints can be defined, resulting in possible errors at execution time.

The *search-condition* "IS NOT NULL" can be specified; however, it is recommended that nullability be enforced directly, using the NOT NULL attribute of a column. For example, `CHECK (salary + bonus > 30000)` is accepted if salary is set to NULL, because CHECK constraints must be either satisfied or unknown, and in this case, salary is unknown. However, `CHECK (salary IS NOT NULL)` would be considered false and a violation of the constraint if salary is set to NULL.

Check constraints with *search-condition* are enforced when rows in the table are inserted or updated. A check constraint defined on a table automatically applies to all subtables of that table.

A functional dependency is not enforced by the database manager during normal operations such as insert, update, delete, or set integrity. The functional dependency might be used during query rewrite to optimize queries. Incorrect results might be returned if the integrity of a functional dependency is not maintained.

constraint-attributes
Defines attributes associated with referential integrity or check constraints.

ENFORCED or NOT ENFORCED
Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete. The default is ENFORCED.

ENFORCED
The constraint is enforced by the database manager. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621). ENFORCED cannot be specified when a referential constraint refers to a nickname (SQLSTATE 428G7).

NOT ENFORCED
The constraint is not enforced by the database manager.

TRUSTED
The data can be trusted to conform to the constraint. TRUSTED must be used only if the data in the table is independently known to conform to the constraint. Query results might be unpredictable if the data does not actually conform to the constraint. This is the default option.

Informational constraints must not be violated at any time. Informational constraints are used in query optimization, as well as the incremental processing of REFRESH IMMEDIATE MQT and staging tables. These processes might produce unpredictable results or incorrect MQT and staging table content if the constraints are violated. For example, the order in which parent-child tables are maintained is important. When you want to add rows to a parent-child table, you must insert rows into the parent table first. To remove rows from a parent-child table, you must delete rows from the child table first. This ensures that there are no orphan rows in the child table at any time. If informational constraints are violated, the incremental maintenance of dependent MQT data and staging table data might be optimized based on the violated informational constraints, producing incorrect data.

NOT TRUSTED
The data cannot be trusted to conform to the constraint. NOT TRUSTED is intended for cases where the data conforms to the constraint for most rows, but it is not independently known that all the rows or future additions will conform to the constraint. If a constraint is NOT TRUSTED and enabled for query optimization, then it will not be used to perform optimizations that depend on the data conforming completely to the constraint. NOT TRUSTED can be specified only for referential integrity constraints (SQLSTATE 42613).

ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION
Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances. The default is ENABLE QUERY OPTIMIZATION.

ENABLE QUERY OPTIMIZATION
The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

OF *type-name1*
Specifies that the columns of the table are based on the attributes of the structured type identified by *type-name1*. If *type-name1* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be an instantiable structured type (SQLSTATE 428DP) with at least one attribute (SQLSTATE 42997).

If UNDER is not specified, an object identifier column must be specified (refer to the *OID-column-definition*). This object identifier column is the first column of the table. The object ID column is followed by columns based on the attributes of *type-name1*.

HIERARCHY *hierarchy-name*
Names the hierarchy table associated with the table hierarchy. It is created at the same time as the root table of the hierarchy. The data for all subtables in the typed table hierarchy is stored in the hierarchy table. A hierarchy table cannot be directly referenced in SQL statements. A *hierarchy-name* is a *table-name*. The *hierarchy-name*, including the implicit or explicit schema name, must not identify a table, nickname, view, or alias described in the catalog. If the schema name is specified, it must be the same as the schema name of the table being created (SQLSTATE 428DQ). If this clause is omitted when defining the root table, a name is generated by the system. This name consists of the name of the table being created, followed by a unique suffix, such that the identifier is unique among the identifiers of existing tables, views, and nicknames.

UNDER *supertable-name*
Indicates that the table is a subtable of *supertable-name*. The supertable must be an existing table (SQLSTATE 42704) and the table must be defined using a structured type that is the immediate supertype of *type-name1* (SQLSTATE 428DB). The schema name of *table-name* and *supertable-name* must be the same (SQLSTATE 428DQ). The table identified by *supertable-name* must not have any existing subtable already defined using *type-name1* (SQLSTATE 42742).

The columns of the table include the object identifier column of the supertable with its type modified to be REF(*type-name1*), followed by columns based on the attributes of *type-name1* (remember that the type includes the attributes of its supertype). The attribute names cannot be the same as the OID column name (SQLSTATE 42711).

Other table options, including table space, data capture, not logged initially, and distribution key options cannot be specified. These options are inherited from the supertable (SQLSTATE 42613).

INHERIT SELECT PRIVILEGES
Any user or group holding a SELECT privilege on the supertable will be granted an equivalent privilege on the newly created subtable. The subtable definer is considered to be the grantor of this privilege.

typed-element-list
Defines the additional elements of a typed table. This includes the additional options for the columns, the addition of an object identifier column (root table only), and constraints on the table.

OID-column-definition
Defines the object identifier column for the typed table.

REF IS *OID-column-name* **USER GENERATED**
Specifies that an object identifier (OID) column is defined in the table as the first column. An OID is required for the root table of a table hierarchy (SQLSTATE 428DX). The table must be a typed table (the OF clause must be present) that is not a subtable (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name1* (SQLSTATE 42711). The column is defined with type REF(*type-name1*), NOT NULL and a system required unique index (with a default index name) is generated. This column is referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

with-options
Defines additional options that apply to columns of a typed table.

column-name
Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of a column of the table that is not also a column of a supertable (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS clause in the statement (SQLSTATE 42613).

If an option is already specified as part of the type definition (in CREATE TYPE), the options specified here override the options in CREATE TYPE.

WITH OPTIONS *column-options*
Defines options for the specified column. See *column-options* described earlier. If the table is a subtable, primary key or unique constraints cannot be specified (SQLSTATE 429B3).

LIKE *table-name1* **or** *view-name* **or** *nickname*
Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name1*), view (*view-name*) or nickname (*nickname*). The name specified after LIKE must identify a table, view or nickname that exists in the catalog, or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC).

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table (including implicitly hidden columns), view, or nickname. A column of the new table that corresponds to an implicitly hidden column in the existing table will also be defined as implicitly hidden. The implicit definition depends on what is identified after LIKE:

- If a table is identified, then the implicit definition includes the column name, data type, hidden attribute, and nullability characteristic of each of the columns of *table-name1*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*. The data types of the view columns must be data types that are valid for columns of a table.
- If a nickname is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each column of *nickname*.
- If a protected table is identified in the LIKE clause, the new table inherits the same security policy and protected columns as the identified table.
- If a table is identified in the LIKE clause and the table contains a row-begin column, row-end column, or transaction-start-ID column, the corresponding column of the new table inherits only the data type of the source column. The new column is not considered a generated column.
- When a table that includes a period is identified in the LIKE clause, the new table does not inherit the period definition.
- When a system-period temporal table is identified in the LIKE clause, the new table is not a system-period temporal table.

Column default and identity column attributes may be included or excluded, based on the copy-attributes clauses. The implicit definition does not include any other attributes of the identified table, view or nickname. Thus the new table does not have any unique constraints, foreign key constraints, triggers, indexes, ORGANIZE BY specification, or PARTITIONING KEY specification. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

When a table is identified in the LIKE clause and that table contains a ROW CHANGE TIMESTAMP column, the corresponding column of the new table inherits only the data type of the ROW CHANGE TIMESTAMP column. The new column is not considered to be a generated column.

If row or column level access control is activated for *table-name-1*, it is not inherited by the new table.

copy-options

These options specify whether or not to copy additional attributes of the source result table definition (table, view or fullselect).

INCLUDING COLUMN DEFAULTS

Column defaults for each updatable column of the source result table definition are copied. Columns that are not updatable will not have a default defined in the corresponding column of the created table.

If LIKE *table-name* is specified and *table-name* identifies a base table, created temporary table, or declared temporary table, then INCLUDING COLUMN DEFAULTS is the default. If LIKE *table-name* is specified and *table-name* identifies a nickname, then INCLUDING COLUMN DEFAULTS has no effect and column defaults are not copied.

EXCLUDING COLUMN DEFAULTS

Columns defaults are not copied from the source result table definition.

This clause is the default, except when LIKE *table-name* is specified and *table-name* identifies a base table, created temporary table, or declared temporary table.

INCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are copied from the source result table definition, if possible. It is possible to copy the identity column attributes, if the element of the corresponding column in the table, view, or fullselect is the name of a table column, or the name of a view column which directly or indirectly maps to the name of a base table column with the identity property. In all other cases, the columns of the new table will not get the identity property. For example:

- the select-list of the fullselect includes multiple instances of an identity column name (that is, selecting the same column more than once)
- the select list of the fullselect includes multiple identity columns (that is, it involves a join)
- the identity column is included in an expression in the select list
- the fullselect includes a set operation (union, except, or intersect).

EXCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are not copied from the source result table definition.

as-result-table

column-name
Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the *fullselect*. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the *fullselect*.

A list of column names must be specified if the result table of the fullselect has duplicate column names of an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS Introduces the query that is used for the definition of the table.

fullselect
Defines the query on which the table is based. The resulting column definitions are the same as those for a view defined with the same query. A column of the new table that corresponds to an implicitly hidden column of a base table referenced in the *fullselect* is not considered hidden in the new table.

Every select list element must have a name (use the AS clause for expressions). The *as-result-table* defines attributes of the table. The data types of the result columns must be data types that are valid for columns of a table.

The *fullselect* cannot include a *data-change-table-reference* clause (SQLSTATE 428FL).

Any valid *fullselect* that does not reference a typed table or a typed view can be specified.

If row or column level access control is activated for tables that are specified in *fullselect*, it is not cascaded to the new table.

WITH NO DATA
The query is used only to define the table. The table is not populated using the results of the query.

The columns of the table are defined based on the definitions of the columns that result from the *fullselect*. If the *fullselect* references a single table in the FROM clause, select list items that are columns of that table are defined using the column name, data type, and nullability characteristic of the referenced table.

materialized-query-definition

column-name
Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the *fullselect* has duplicate column names of an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS Introduces the query that is used for the definition of the table and that determines the data to be included in the table.

fullselect
Defines the query on which the table is based. The resulting column definitions are the same as those for a view defined with the same query. A column of the new table that corresponds to an implicitly hidden column of a base table referenced in the fullselect is not considered hidden in the new table.

Every select list element must have a name (use the AS clause for expressions). The *materialized-query-definition* defines attributes of the materialized query table. The option chosen also defines the contents of the fullselect as follows.

The fullselect cannot include a *data-change-table-reference* clause (SQLSTATE 428FL).

When REFRESH DEFERRED or REFRESH IMMEDIATE is specified, the fullselect cannot include (SQLSTATE 428EC):

- References to a materialized query table, created temporary table, declared temporary table, or typed table in any FROM clause
- References to a view where the fullselect of the view violates any of the listed restrictions on the fullselect of the materialized query table
- Expressions that are a reference type (or distinct type based on this type)
- Functions that have any of the following attributes:
  - EXTERNAL ACTION
  - LANGUAGE SQL
  - CONTAINS SQL
  - READS SQL DATA
  - MODIFIES SQL DATA

- NOT SECURED functions if the functions reference a materialized query table which then references a table that has row or column access control activated.
- Functions that depend on physical characteristics (for example, DBPARTITIONNUM, HASHEDVALUE, RID_BIT, RID)
- A ROW CHANGE expression or reference to a ROW CHANGE TIMESTAMP column of the row
- Table or view references to system objects (Explain tables also should not be specified)
- Expressions that are a structured type, LOB type (or a distinct type based on a LOB type), or XML type
- References to a protected table or protected nickname

When DISTRIBUTE BY REPLICATION is specified, the following restrictions apply:

- The GROUP BY clause is not allowed.
- The materialized query table must only reference a single table; that is, it cannot include a join.

When REFRESH IMMEDIATE is specified:

- The query must be a subselect, with the exception that UNION ALL is supported in the input table expression of a GROUP BY.
- The query cannot be recursive.
- The query cannot include:
  - References to a nickname
  - Functions that are not deterministic
  - Scalar fullselects
  - Predicates with fullselects
  - Special registers and built-in functions that depend on the value of a special register
  - Global variables
  - SELECT DISTINCT
  - An error tolerant *nested-table-expression*
- If the FROM clause references more than one table or view, it can only define an inner join without using the explicit INNER JOIN syntax.
- When a GROUP BY clause is specified, the following considerations apply:
  - The supported column functions are SUM, COUNT, COUNT_BIG and GROUPING (without DISTINCT). The select list must contain a COUNT(*) or COUNT_BIG(*) column. If the materialized query table select list contains SUM(X), where X is a nullable argument, the materialized query table must also have COUNT(X) in its select list. These column functions cannot be part of any expressions.
  - A HAVING clause is not allowed.
  - If in a multiple partition database partition group, the distribution key must be a subset of the GROUP BY items.
- The materialized query table must not contain duplicate rows, and the following restrictions specific to this uniqueness requirement apply, depending upon whether or not a GROUP BY clause is specified.
  - When a GROUP BY clause is specified, the following uniqueness-related restrictions apply:

- All GROUP BY items must be included in the select list.
- When the GROUP BY contains GROUPING SETS, CUBE, or ROLLUP, the GROUP BY items and associated GROUPING column functions in the select list must form a unique key of the result set. Thus, the following restrictions must be satisfied:
  - No grouping sets can be repeated. For example, `ROLLUP(X,Y),X` is not allowed, because it is equivalent to `GROUPING SETS((X,Y),(X),(X))`.
  - If X is a nullable GROUP BY item that appears within GROUPING SETS, CUBE, or ROLLUP, then GROUPING(X) must appear in the select list.

– When a GROUP BY clause is not specified, the following uniqueness-related restrictions apply:
- The materialized query table's uniqueness requirement is achieved by deriving a unique key for the materialized view from one of the unique key constraints defined in each of the underlying tables. Therefore, the underlying tables must have at least one unique key constraint defined on them, and the columns of these keys must appear in the select list of the materialized query table definition.

When REFRESH DEFERRED is specified:

- If the materialized query table is created with the intention of providing it with an associated staging table in a later statement, the fullselect of the materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.
- If the query is recursive, the materialized query table is not used to optimize the processing of queries.
- The materialized query table is not used to optimize the processing of static queries.

A materialized query table whose fullselect contains a GROUP BY clause is summarizing data from the tables referenced in the fullselect. Such a materialized query table is also known as a *summary table*. A summary table is a specialized type of materialized query table.

If the *fullselect* references a table or a view that depends on a table for which row or column level access control has been activated, those row or column level access controls are ignored when populating the materialized query table. The materialized query table is automatically created with row level access control activated. Direct access by end users to this table does not see any content unless appropriate permissions are created or a user with SECADM authority chooses to deactivate row level access control on this materialized query table. Note that row and column level access control on the materialized query table does not affect internal routing by the DB2 SQL compiler to the materialized query table.

refreshable-table-options

Define the refreshable options of the materialized query table attributes.

DATA INITIALLY DEFERRED

Data is not inserted into the table as part of the CREATE TABLE statement. A REFRESH TABLE statement specifying the *table-name* is used to insert data into the table.

REFRESH

Indicates how the data in the table is maintained.

DEFERRED
The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. System-maintained materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807). User-maintained materialized query tables defined with this attribute do allow INSERT, UPDATE, or DELETE statements.

IMMEDIATE
The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified *subselect* is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION
The materialized query table can be used for query optimization under appropriate circumstances.

DISABLE QUERY OPTIMIZATION
The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY
Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool. The default is SYSTEM.

SYSTEM
Specifies that the data in the materialized query table is maintained by the system.

USER
Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL
Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

When specifying this option, the select clause in the CREATE TABLE statement cannot contain a reference to a base table (SQLSTATE 428EC).

staging-table-definition
Defines the query supported by the staging table indirectly through an associated materialized query table. The underlying tables of the materialized

query table are also the underlying tables for its associated staging table. The staging table collects changes that need to be applied to the materialized query table to synchronize it with the contents of the underlying tables.

If the *fullselect* references a table or a view that depends on a table for which row or column level access control has been activated, those row or column level access controls are ignored when populating the staging table. However, the staging table is automatically created with row level access control activated. Direct access by end users to this staging table does not see any content unless appropriate permissions are created or a user with SECADM authority chooses to deactivate row level access control on this staging table. Note that row and column level access control on the staging table does not affect the internal process of applying the changes captured by the staging table to the associated materialized query table.

staging-column-name
Names the columns in the staging table. If a list of column names is specified, it must consist of *two* more names than there are columns in the materialized query table for which the staging table is defined. If the materialized query table is a replicated materialized query table, or the query defining the materialized query table does not contain a GROUP BY clause, the list of column names must consist of *three* more names than there are columns in the materialized query table for which the staging table is defined. Each column name must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the associated materialized query table. The additional columns are named GLOBALTRANSID and GLOBALTRANSTIME, and if a third column is necessary, it is named OPERATIONTYPE.

Table 20. Extra Columns Appended in Staging Tables

Column Name	Data Type	Column Description
GLOBALTRANSID	CHAR(8) FOR BIT DATA	The global transaction ID for each propagated row
GLOBALTRANSTIME	CHAR(13) FOR BIT DATA	The timestamp of the transaction
OPERATIONTYPE	INTEGER	Operation for the propagated row, either insert, update, or delete.

A list of column names must be specified if any of the columns of the associated materialized query table duplicates any of the generated column names (SQLSTATE 42711).

FOR *table-name2*
Specifies the materialized query table that is used for the definition of the staging table. The name, including the implicit or explicit schema, must identify a materialized query table that exists at the current server defined with REFRESH DEFERRED. The fullselect of the associated materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.

The contents of the staging table can be used to refresh the materialized query table, by invoking the REFRESH TABLE statement, if the contents of the staging table are consistent with the associated materialized query table and the underlying source tables.

PROPAGATE IMMEDIATE
The changes made to the underlying tables as part of a delete, insert, or update operation are cascaded to the staging table in the same delete, insert, or update operation. If the staging table is not marked inconsistent, its content, at any point-in-time, is the delta changes to the underlying table since the last refresh materialized query table.

ORGANIZE BY DIMENSIONS (*column-name***,...)**
Specifies a dimension for each column or group of columns used to cluster the table data. The use of parentheses within the dimension list specifies that a group of columns is to be treated as one dimension. The DIMENSIONS keyword is optional. A table whose definition specifies this clause is known as a multidimensional clustering (MDC) table.

A clustering block index is automatically maintained for each specified dimension, and a block index, consisting of all columns used in the clause, is maintained if none of the clustering block indexes includes them all. The set of columns used in the ORGANIZE BY clause must follow the rules for the CREATE INDEX statement that specifies CLUSTER.

Each column name specified in the ORGANIZE BY clause must be defined for the table (SQLSTATE 42703). A dimension cannot occur more than once in the dimension list (SQLSTATE 42709). The dimensions cannot contain a ROW CHANGE TIMESTAMP column, row-begin column, row-end column, transaction-start-ID column (SQLSTATE 429BV), or an XML column (SQLSTATE 42962).

Pages of the table are arranged in blocks of equal size, which is the extent size of the table space, and all rows of each block contain the same combination of dimension values.

A table can be both a multidimensional clustering (MDC) table and a partitioned table. Columns in such a table can be used in both the *range-partition-spec* and in the MDC key. Note that table partitioning is multi-column, not multidimensional.

For a partitioned MDC table created by DB2 Version 9.7 Fix Pack 1 or later releases, the block indexes are partitioned. The partitioned block index placement follows the general partitioned index storage placement rule. All index partitions for a given data partition, including MDC block indexes, share a single index object. By default, the index partitions for each specific data partition reside in the same table space as the data partition. This can be overridden with the partition level INDEX IN clause.

For MDC tables created by DB2 V9.7 or earlier releases, the block indexes are nonpartitioned and remain nonpartitioned if they are rebuilt. MDC tables with partitioned block indexes can co-exist in the same database as MDC tables with nonpartitioned block indexes. To change nonpartitioned block indexes to partitioned block indexes, use an online table move to migrate the MDC table.

ORGANIZE BY KEY SEQUENCE *sequence-key-spec*
Specifies that the table is organized in ascending key sequence with a fixed size based on the specified range of key sequence values. A table organized in this way is referred to as a *range-clustered table*. Each possible key value in the defined range has a predetermined location in the physical table. The storage required for a range-clustered table must be available when the table is created, and must be sufficient to contain the number of rows in the specified range multiplied by the row size (for details on determining the space requirement, see Row Size Limit and Byte Counts).

column-name
Specifies a column of the table that is included in the unique key that determines the sequence of the range-clustered table. The data type of the column must be SMALLINT, INTEGER, or BIGINT (SQLSTATE 42611), and the columns must be defined as NOT NULL (SQLSTATE 42831). The same column must not be identified more than once in the sequence key. The number of identified columns must not exceed 64 (SQLSTATE 54008).

A unique index entry will automatically be created in the catalog for the columns in the key sequence specified with ascending order for each column. The name of the index will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name. An actual index object is not created in storage, because the table organization is ordered by this key. If a primary key or a unique constraint is defined on the same columns as the range-clustered table sequence key, this same index entry is used for the constraint.

For the key sequence specification, a check constraint exists to reflect the column constraints. If the DISALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT, and the check constraint is enforced. If the ALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT_OFLOW, and the check constraint is not enforced.

STARTING FROM *constant*
Specifies the constant value at the low end of the range for *column-name*. Values less than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821). If a starting constant is not specified, the default value is 1.

ENDING AT *constant*
Specifies the constant value at the high end of the range for *column-name*. Values greater than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. The value of the ending constant must be greater than the starting constant. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821).

ALLOW OVERFLOW
Specifies that the range-clustered table allows rows with key values that are outside of the defined range of values. When a range-clustered table is created to allow overflows, the rows with key values outside of the range are placed at the end of the defined range without any predetermined order. Operations involving these overflow rows are less efficient than operations on rows having key values within the defined range.

DISALLOW OVERFLOW
Specifies that the range-clustered table does not allow rows with key values that are not within the defined range of values (SQLSTATE 23513). Range-clustered tables that disallow overflows will always maintain all rows in ascending key sequence.

The DISALLOW OVERFLOW clause cannot be specified if the table is a range-clustered materialized query table (SQLSTATE 429BG).

PCTFREE *integer*
Specifies the percentage of each page that is to be left as free space. The first row on each page is added without restriction. When additional rows are added to a page, at least *integer* percent of the page is left as free space. The value of *integer* can range from 0 to 99. A PCTFREE value of -1 in the system catalog (SYSCAT.TABLES) is interpreted as the default value. The default PCTFREE value for a table page is 0.

ORGANIZE BY INSERT TIME
Specifies that rows are clustered in the table relative to the time they are inserted. Rows are inserted at the logical end of the table object instead of searching for available space. A table which is organized by insert time is known as an insert time clustering (ITC) table. This type of table can use REORG TABLE RECLAIM EXTENTS to reclaim free extents for immediate use by other objects in the table space.

Data is clustered using an implicitly created virtual dimension. A clustering block index is automatically maintained for this virtual dimension. The virtual dimension cannot be manipulated and it consumes no space for each row that exists in the table. Pages of the table are arranged in blocks of equal size, which is the extent size of the table space.

The ORGANIZE BY INSERT TIME clause cannot be specified if the table is a typed table (SQLSTATE 428DH).

DATA CAPTURE
Indicates whether extra information for inter-database data replication is to be written to the log. This clause cannot be specified when creating a subtable (SQLSTATE 428DR).

If the clause is not specified and that table is not a typed table, then the default is determined by the DATA CAPTURE setting of the schema at the time the table is created.

NONE
Indicates that no extra information will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the schema name (implicit or explicit) of the table is longer than 18 bytes, this option is not supported (SQLSTATE 42997).

If the table is a typed table that is not a subtable, then this option is not supported (SQLSTATE 428DH).

IN *tablespace-name*,...
Identifies the table spaces in which the table will be created. The table spaces must exist, they must be in the same database partition group, and they must be all regular DMS or all large DMS or all SMS table spaces (SQLSTATE 42838) on which the authorization ID of the statement holds the USE privilege.

A maximum of one IN clause is allowed at the table level. All data table spaces used by a table must have the same page size and extent size.

If only one table space is specified, all table parts are stored in this table space. This clause cannot be specified when creating a subtable (SQLSTATE 42613), because the table space is inherited from the root table of the table hierarchy.

If this clause is not specified, the database manager chooses a table space (from the set of existing table spaces in the database) with the smallest sufficient page size on which the authorization ID of the statement has USE privilege.

If more than one table space qualifies, choose the table space in the following order of preference, depending how the authorization ID of the statement was granted USE privilege on the table space:

1. The authorization ID
2. A role to which the authorization ID is granted
3. A group to which the authorization ID belongs
4. A role to which a group the authorization ID belongs is granted
5. PUBLIC
6. A role to which PUBLIC is granted

If more than one table space still qualifies, the final choice is made by the database manager.

Table space determination can change if:

- Table spaces are dropped or created
- USE privileges are granted or revoked

Partitioned tables can have their data partitions spread across multiple table spaces. When multiple table spaces are specified, all of the table spaces must exist, and they must all be either SMS or regular DMS or large DMS table spaces (SQLSTATE 42838). The authorization ID of the statement must hold the USE privilege on all of the specified table spaces.

The sufficient page size of a table is determined by either the byte count of the row or the number of columns. For more information, see Row Size Limits.

When a table is placed in a large table space:

- The table can be larger than a table in a regular table space. For details on table and table space limits, see “SQL limits”.
- The table can support more than 255 rows per data page, which can improve space utilization on data pages.
- Indexes that are defined on the table will require an additional 2 bytes per row entry, compared to indexes defined on a table that resides in a regular table space.

CYCLE or NO CYCLE
Specifies whether or not the number of data partitions with no explicit table space can exceed the number of specified table spaces.

CYCLE
Specifies that if the number of data partitions with no explicit table space exceeds the number of specified table spaces, the table spaces are assigned to data partitions in a round-robin fashion.

NO CYCLE
Specifies that the number of data partitions with no explicit table space must not exceed the number of specified tables spaces (SQLSTATE 428G1). This option prevents the round-robin assignment of table spaces to data partitions.

tablespace-options
Specifies the table space in which indexes or long column values are to be stored. For details on types of table spaces, see “CREATE TABLESPACE”.

INDEX IN *tablespace-name*
Identifies the table space in which any indexes on a nonpartitioned table or nonpartitioned indexes on a partitioned table are to be created. The specified table space must exist; it must be a DMS table space if the table has data in DMS table spaces, or an SMS table space if the partitioned table has data in SMS table spaces; it must be a table space on which the authorization ID of the statement holds the USE privilege; and it must be in the same database partition group as *tablespace-name* (SQLSTATE 42838).

Specifying which table space will contain indexes can be done when a table is created or, in the case of partitioned tables, it can be done by specifying the IN clause of the CREATE INDEX statement for a nonpartitioned index. Checking for the USE privilege on the table space is done at table creation time, not when an index is created later.

For a nonpartitioned index on a partitioned table, storage of the index is as follows:

- The table space by the IN clause of the CREATE INDEX statement
- The table-level table space specified for the INDEX IN clause of the CREATE TABLE statement
- If neither of the preceding are specified, the index is stored in the table space of the first attached or visible data partition

For information about partitioned indexes on partitioned tables, see the description of the partition-element INDEX IN clause.

LONG IN *tablespace-name*
Identifies the table spaces in which the values of any long columns are to be stored. Long columns include those with LOB data types, XML type, distinct types with any of these as source types, or any columns defined with user-defined structured types whose values cannot be stored inline. This option is allowed only if the IN clause identifies a DMS table space.

The specified table space must exist. It can be a regular table space if it is the same table space in which the data is stored; otherwise, it must be a large DMS table space on which the authorization ID of the statement holds the USE privilege. It must also be in the same database partition group as *tablespace-name* (SQLSTATE 42838).

Specifying which table space will contain long, LOB, or XML columns can only be done when a table is created. Checking for the USE privilege is done at table creation time, not when a long or LOB column is added later.

For rules governing the use of the LONG IN clause with partitioned tables, see "Large object behavior in partitioned tables".

distribution-clause
Specifies the database partitioning or the way the data is distributed across multiple database partitions.

DISTRIBUTE BY HASH (*column-name***,...)**
Specifies the use of the default hashing function on the specified columns, called a *distribution key*, as the distribution method across database partitions. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column whose data type is BLOB, CLOB, DBCLOB, XML, distinct type based on any of these types,

or structured type can be used as part of a distribution key (SQLSTATE 42962). The distribution key cannot contain a ROW CHANGE TIMESTAMP column (SQLSTATE 429BV). A distribution key cannot be specified for a table that is a subtable (SQLSTATE 42613), because the distribution key is inherited from the root table in the table hierarchy or a table with a column of data type XML (SQLSTATE 42997). A distribution key cannot contain row begin/row end/transaction start id columns. If this clause is not specified, and the table resides in a multiple partition database partition group with multiple database partitions, the distribution key is defined as follows:

- If the table is a typed table, the object identifier column is the distribution key.
- If a primary key is defined, the first column of the primary key is the distribution key.
- Otherwise, the first column whose data type is valid for a distribution key becomes the distribution key.

The columns of the distribution key must be a subset of the columns that make up any unique constraints.

If none of the columns satisfies the requirements for a default distribution key, the table is created without one. Such tables are allowed only in table spaces that are defined on single-partition database partition groups.

For tables in table spaces that are defined on single-partition database partition groups, any collection of columns with data types that are valid for a distribution key can be used to define the distribution key. If you do not specify this clause, no distribution key is created.

For restrictions related to the distribution key, see Rules.

DISTRIBUTE BY REPLICATION
Specifies that the data stored in the table is physically replicated on each database partition of the database partition group for the table spaces in which the table is defined. This means that a copy of all of the data in the table exists on each database partition. This option can only be specified for a materialized query table (SQLSTATE 42997).

partitioning-clause
Specifies how the data is partitioned within a database partition.

PARTITION BY RANGE *range-partition-spec*
Specifies the table partitioning scheme for the table.

partition-expression
Specifies the key data over which the range is defined to determine the target data partition of the data.

column-name
Identifies a column of the table-partitioning key. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column with a data type that is a BLOB, CLOB, DBCLOB, XML, distinct type based on any of these types, or structured type can be used as part of a table-partitioning key (SQLSTATE 42962).

The numeric literals used in the range specification are governed by the rules for numeric literals. All of the numeric literals (except the decimal floating-point special values) used in ranges

corresponding to numeric columns are interpreted as integer, floating-point or decimal constants, in accordance with the rules specified for numeric constants. As a result, for decimal floating-point columns, the minimum and maximum numeric constant value that can be used in the range specification of a data partition is the smallest DOUBLE value and the largest DOUBLE value, respectively. Decimal floating-point special values can be used in the range specification. All decimal floating-point special values are interpreted as greater than MINVALUE and less than MAXVALUE.

The table partitioning columns cannot contain a ROW CHANGE TIMESTAMP column (SQLSTATE 429BV). The number of identified columns must not exceed 16 (SQLSTATE 54008).

NULLS LAST
Indicates that null values compare high.

NULLS FIRST
Indicates that null values compare low.

partition-element
Specifies ranges for a data partitioning key and the table space where rows of the table in the range will be stored.

PARTITION *partition-name*
Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec
Specifies the boundaries of a data partition. The lowest data partition must include a starting-clause, and the highest data partition must include an ending-clause (SQLSTATE 56016). Data partitions between the lowest and the highest can include either a starting-clause, ending-clause, or both clauses. If only the ending-clause is specified, the previous data partition must also have included an ending-clause (SQLSTATE 56016).

starting-clause
Specifies the low end of the range for a data partition. There must be at least one starting value specified and no more values than the number of columns in the data partitioning key (SQLSTATE 53038). If there are fewer values specified than the number of columns, the remaining values are implicitly MINVALUE.

STARTING FROM
Introduces the *starting-clause*.

constant
Specifies a constant value with a data type that is assignable to the data type of the *column-name* to which it corresponds (SQLSTATE 53045). The value must not be in the range of any other boundary-spec for the table (SQLSTATE 56016).

MINVALUE
Specifies a value that is lower than the lowest possible value for the data type of the *column-name* to which it corresponds.

MAXVALUE
Specifies a value that is greater than the greatest possible value for the data type of the *column-name* to which it corresponds.

INCLUSIVE
Indicates that the specified range values are to be included in the data partition.

EXCLUSIVE
Indicates that the specified *constant* values are to be excluded from the data partition. This specification is ignored when MINVALUE or MAXVALUE is specified.

ending-clause
Specifies the high end of the range for a data partition. There must be at least one starting value specified and no more values than the number of columns in the data partitioning key (SQLSTATE 53038). If there are fewer values specified than the number of columns, the remaining values are implicitly MAXVALUE.

ENDING AT
Introduces the *ending-clause*.

constant
Specifies a constant value with a data type that is assignable to the data type of the *column-name* to which it corresponds (SQLSTATE 53045). The value must not be in the range of any other boundary-spec for the table (SQLSTATE 56016).

MINVALUE
Specifies a value that is lower than the lowest possible value for the data type of the *column-name* to which it corresponds.

MAXVALUE
Specifies a value that is greater than the greatest possible value for the data type of the *column-name* to which it corresponds.

INCLUSIVE
Indicates that the specified range values are to be included in the data partition.

EXCLUSIVE
Indicates that the specified *constant* values are to be excluded from the data partition. This specification is ignored when MINVALUE or MAXVALUE is specified.

IN *tablespace-name*
Specifies the table space where the data partition is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE

42838); it must be a table space on which the authorization ID of the statement holds the USE privilege. If this clause is not specified, a table space is assigned by default in a round-robin fashion from the list of table spaces specified for the table. If a table space was not specified for large objects using the LONG IN clause, large objects are placed in the same table space as are the rest of the rows for the data partition. For partitioned tables, the LONG IN clause can be used to provide a list of table spaces. This list is used in round robin-fashion to place large objects for each data partition. For rules governing the use of the LONG IN clause with partitioned tables, see "Large object behavior in partitioned tables".

If the INDEX IN clause is not specified on the CREATE TABLE or the CREATE INDEX statement, the index is placed in the same table space as the first visible or attached partition of the table.

INDEX IN *tablespace-name*
Specifies the table space where the partitioned index on the partitioned table is to be stored.

The partition-element level INDEX IN clause only affects the storage of partitioned indexes. Storage of the index is as follows:

- If the INDEX IN clause is specified at the partition level when the table is created, the partitioned index is stored in the specified table space.
- If the INDEX IN clause is not specified at the partition level when the table is created, the partitioned index is stored in the table space of the corresponding data partition.

The INDEX IN clause can only be specified if the data table spaces are DMS table spaces and the table space specified by the INDEX IN clause is a DMS table space. If the data table space is an SMS table space, an error is returned (SQLSTATE 42839).

LONG IN *tablespace-name*
Identifies the table spaces in which the values of any long columns are to be stored. Long columns include those with LOB data types, XML type, distinct types with any of these as source types, or any columns defined with user-defined structured types whose values cannot be stored inline. This option is allowed only if the IN clause identifies a DMS table space.

The specified table space must exist. It can be a regular table space if it is the same table space in which the data is stored; otherwise, it must be a large DMS table space on which the authorization ID of the statement holds the USE privilege. It must also be in the same database partition group as *tablespace-name* (SQLSTATE 42838).

Specifying which table space will contain long, LOB, or XML columns can only be done when a table is created. Checking for the USE privilege is done at table creation time, not when a long or LOB column is added later.

For rules governing the use of the LONG IN clause with partitioned tables, see "Large object behavior in partitioned tables".

EVERY (*constant***)**
Specifies the width of each data partition range when using the

automatically generated form of the syntax. Data partitions will be created starting at the STARTING FROM value and containing this number of values in the range. This form of the syntax is only supported for tables that are partitioned by a single numeric or datetime column (SQLSTATE 53038).

If the partitioning key column is a numeric type, the starting value of the first partition is the value specified in the starting-clause. The ending value for the first and all other partitions is calculated by adding the starting value of the partition to the increment value specified as *constant* in the EVERY clause. The starting value for all other partitions is calculated by taking the starting value for the previous partition and adding the increment value specified as *constant* in the EVERY clause.

If the partitioning key column is a DATE or a TIMESTAMP, the starting value of the first partition is the value specified in the starting-clause. The ending value for the first and all other partitions is calculated by adding the starting value of the partition to the increment value specified as a labeled duration in the EVERY clause. The starting value for all other partitions is calculated by taking the starting value for the previous partition and adding the increment value specified as a labeled duration in the EVERY clause.

For a numeric column, the EVERY value must be a positive numeric constant, and for a datetime column, the EVERY value must be a labeled duration (SQLSTATE 53045).

COMPRESS

Specifies whether data compression applies to the rows of the table

NO Specifies that data row compression is disabled.

YES

Specifies that data row compression is enabled. Insert and update operations on the table will be subject to compression. Any XML storage objects that exist are also compressed. For both adaptive and classic row compression, a table-level compression dictionary is automatically created after the table is sufficiently populated with data. This also applies to the data in the XML storage object; if there is sufficient data in the XML storage object, a compression dictionary is automatically created and XML documents are subject to compression.

Note: The compression applied to the XML storage object is the same, regardless of whether you use adaptive or classic row compression.

For adaptive row compression, page-level compression dictionaries are created or updated as soon as data is inserted or changed in the table.

ADAPTIVE

Enables adaptive compression, and records are subject to being compressed with a table-level and a page-level compression dictionary. This is the default option when COMPRESS YES is specified. The functionality of COMPRESS YES ADAPTIVE is a superset of the functionality of COMPRESS YES STATIC.

STATIC
Enables classic row compression using a table-level compression dictionary. This is the same row compression functionality that existed in previous DB2 versions.

VALUE COMPRESSION
This determines the row format that is to be used. Each data type has a different byte count depending on the row format that is used. For more information, see Byte Counts. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

The null value is stored using three bytes. This is the same or less space than when VALUE COMPRESSION is not active for columns of all data types, with the exception of CHAR(1). Whether or not a column is defined as nullable has no affect on the row size calculation. The zero-length data values for columns whose data type is VARCHAR, VARGRAPHIC, LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, BLOB, or XML are to be stored using two bytes only, which is less than the storage required when VALUE COMPRESSION is not active. When a column is defined using the COMPRESS SYSTEM DEFAULT option, this also allows the system default value for the column to be stored using three bytes of total storage. The row format that is used to support this determines the byte counts for each data type, and tends to cause data fragmentation when updating to or from the null value, a zero-length value, or the system default value.

WITH RESTRICT ON DROP
Indicates that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

NOT LOGGED INITIALLY
Any changes made to the table by an Insert, Delete, Update, Create Index, Drop Index, or Alter Table operation in the same unit of work in which the table is created are not logged. For other considerations when using this option, see the "Notes" section of this statement.

All catalog changes and storage related information are logged, as are all operations that are done on the table in subsequent units of work.

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back (SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY attribute was activated is marked inaccessible after the rollback has occurred, and can only be dropped. Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

CCSID
Specifies the encoding scheme for string data stored in the table. If the CCSID clause is not specified, the default is CCSID UNICODE for Unicode databases, and CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE
Specifies that string data is encoded in Unicode. If the database is a

Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8.

If the database is not a Unicode database, tables can be created with CCSID UNICODE, but the following rules apply:

- The alternate collating sequence must be specified in the database configuration before creating the table (SQLSTATE 56031). CCSID UNICODE tables collate with the alternate collating sequence specified in the database configuration.
- Tables or table functions created with CCSID ASCII, and tables or table functions created with CCSID UNICODE, cannot both be used in a single SQL statement (SQLSTATE 53090). This applies to tables and table functions referenced directly in the statement, as well as to tables and table functions referenced indirectly (such as, for example, through referential integrity constraints, triggers, materialized query tables, and tables in the body of views).
- Tables created with CCSID UNICODE cannot be referenced in SQL functions or SQL methods (SQLSTATE 560C0).
- An SQL statement that references a table created with CCSID UNICODE cannot invoke an SQL function or SQL method (SQLSTATE 53090).
- Graphic types, the XML type, and user-defined types cannot be used in CCSID UNICODE tables (SQLSTATE 560C1).
- Anchored data types cannot anchor to columns of a table created with CCSID UNICODE (SQLSTATE 428HS).
- Tables cannot have both the CCSID UNICODE clause and the DATA CAPTURE CHANGES clause specified (SQLSTATE 42613).
- The Explain tables cannot be created with CCSID UNICODE (SQLSTATE 55002).
- Created temporary tables and declared temporary tables cannot be created with CCSID UNICODE (SQLSTATE 56031).
- CCSID UNICODE tables cannot be created in a CREATE SCHEMA statement (SQLSTATE 53090).
- The exception table for a load operation must have the same CCSID as the target table for the operation (SQLSTATE 428A5).
- The exception table for a SET INTEGRITY statement must have the same CCSID as the target table for the statement (SQLSTATE 53090).
- The target table for event monitor data must not be declared as CCSID UNICODE (SQLSTATE 55049).
- Statements that reference a CCSID UNICODE table can only be invoked from a DB2 Version 8.1 or later client (SQLSTATE 42997).
- SQL statements are always interpreted in the database code page. In particular, this means that every character in literals, hex literals, and delimited identifiers must have a representation in the database code page; otherwise, the character will be replaced with the substitution character.

Host variables in the application are always in the application code page, regardless of the CCSID of any tables in the SQL statements that are invoked. DB2 will perform code page conversions as necessary to convert data between the application code page and the section code page. The registry variable DB2CODEPAGE can be set at the client to change the application code page.

SECURITY POLICY
Names the security policy to be associated with the table.

policy-name
Identifies a security policy that already exists at the current server (SQLSTATE 42704).

OPTIONS (*table-option-name string-constant***, ...)**
Table options are used to identify the remote base table. The *table-option-name* is the name of the option. The *string-constant* specifies the setting for the table option. The *string-constant* must be enclosed in single quotation marks.

The remote server (the server name that was specified in the CREATE SERVER statement) must be specified in the OPTIONS clause. The OPTIONS clause can also be used to override the schema or the unqualified name of the remote base table that is being created.

It is recommended that a schema name be specified. If a remote schema name is not specified, the qualifier for the table name is used. If the table name has no qualifier, the authorization ID of the statement is used.

If an unqualified name for the remote base table is not specified, *table-name* is used.

Rules

- The sum of the byte counts of the columns, including the inline lengths of all structured or XML type columns, must not be greater than the row size limit that is based on the page size of the table space (SQLSTATE 54010). For more information, see Byte Counts. For typed tables, the byte count is applied to the columns of the root table of the table hierarchy, and every additional column introduced by every subtable in the table hierarchy (additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable). There is also an additional 4 bytes of overhead to identify the subtable to which each row belongs.
- The number of columns in a table cannot exceed 1 012 (SQLSTATE 54011). For typed tables, the total number of attributes of the types of all of the subtables in the table hierarchy cannot exceed 1010.
- An object identifier column of a typed table cannot be updated (SQLSTATE 42808).
- Any unique or primary key constraint defined on the table must be a superset of the distribution key (SQLSTATE 42997).
- The following rules only apply to multiple database partition databases.
  - Tables composed only of columns with types LOB, XML, a distinct type based on one of these types, or a structured type can only be created in table spaces that are defined on single-partition database partition groups.
  - The distribution key definition of a table in a table space that is defined on a multiple partition database partition group cannot be altered.
  - The distribution key column of a typed table must be the OID column.
  - Partitioned staging tables are not supported.
- For databases running in a DB2 pureScale environment, the ORGANIZE BY clause cannot be specified (SQLSTATE 42997).
- The following restrictions apply to range-clustered tables:
  - A range-clustered table cannot be specified in a DB2 pureScale environment (SQLSTATE 42997).
  - A clustering index cannot be created.

  - Altering the table to add a column is not supported.
  - Altering the table to change the data type of a column is not supported.
  - Altering the table to change PCTFREE is not supported.
  - Altering the table to set APPEND ON is not supported.
  - DETAILED statistics are not available.
  - The load utility cannot be used to populate the table.
  - Columns cannot be of type XML.
- A table is not protected unless it has a security policy associated with it and it includes either a column of type DB2SECURITYLABEL or a column defined with the SECURED WITH clause. The former indicates that the table is a protected table with **row level granularity** and the latter indicates that the table a protected table with **column level granularity**.
- Declaring a column of type DB2SECURITYLABEL fails if the table does not have a security policy associated with it (SQLSTATE 55064).
- A security policy cannot be added to a typed table (SQLSTATE 428DH), materialized query table, or staging table (SQLSTATE 428FG).
- An error tolerant *nested-table-expression* cannot be specified in the fullselect of a *materialized-query-definition* (SQLSTATE 428GG).
- When creating a materialized query table and any of the base tables it depends upon are protected with label-based access control, the following rules apply:
  - Row level security
    - Only one table in the materialized query table's fullselect can have a column type of DB2SECURITYLABEL (SQLSTATE 428FG).
    - The row security label column must be selected and referenced as a stand alone column in the outermost SELECT list in the materialized query table definition (SQLSTATE 428FG). The corresponding column in the materialized query table will be marked as the row security label column.
  - Column level security
    - If a table involved in the materialized query table definition has a column protected with a security label, and that column appears in the materialized query table definition, that column's security label is inherited by the corresponding column in the materialized query table. See the examples in this topic for more details.
  - When creating a materialized query table that depends on one or more tables protected by label-based access control, all base tables must have the same security policy object (SQLSTATE, 428FG). The materialized query table will be automatically protected with that security policy object.
  - The security label associated with a materialized query table column is computed as the aggregate of one or more security labels. This aggregate consists of the security labels associated with the base tables' columns that participate in the definition of that materialized query table column. The aggregate also consists of the security labels associated with any base table columns that appear in other parts of the materialized query table definition, such as the `WHERE`, `ORDER BY`, and `HAVING` clauses. The **`ALTER SECURITY POLICY`** has a description of how two security labels are aggregated. See the examples in this topic for more details.
  - When a staging table is created for a materialized query table that is protected with label-based access control, that staging table carries automatic protection like the materialized query table. See the examples in this topic for more details.

  - Label-based access control is enforced for direct access to a materialized query table just as it is enforced for a regular table. There are no differences from this perspective. When the SQL compiler services a query through a materialized query table, the label-based access control defined on the materialized query table itself does not need to be enforced. The SQL compiler uses the materialized query table which factors in the label-based access control rules from the appropriate base tables.
- The *isolation-clause* cannot be specified in the *full-select* of the *materialized-query-table-definition* (SQLSTATE 42601).
- Subselect statements containing a *lock-request-clause* are not be eligible for MQT routing.
- National character spellings for the graphic data types can be specified only in a Unicode database (SQLSTATE 560AA).
- The following restrictions apply to insert time clustering (ITC) tables:
  - ITC tables are not supported in an SMS table space (SQLSTATE 42838).
  - Indexes defined on ITC tables are not supported in an SMS table space (SQLSTATE 42838).

Notes

- Creating a table with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- If a foreign key is specified:
  - All packages with a delete usage on the parent table are invalidated.
  - All packages with an update usage on at least one column in the parent key are invalidated.
- Creating a subtable causes invalidation of all packages that depend on any table in table hierarchy.
- VARCHAR and VARGRAPHIC columns that are greater than 4 000 and 2 000 respectively should not be used as input parameters in functions in SYSFUN schema. Errors will occur when the function is invoked with an argument value that exceeds these lengths (SQLSTATE 22001).
- The use of NO ACTION or RESTRICT as delete or update rules for referential constraints determines when the constraint is enforced. A delete or update rule of RESTRICT is enforced *before* all other constraints, including those referential constraints with modifying rules such as CASCADE or SET NULL. A delete or update rule of NO ACTION is enforced *after* other referential constraints. One example where different behavior is evident involves the deletion of rows from a view that is defined as a UNION ALL of related tables.

  ```
  Table T1 is a parent of table T3; delete rule as noted below.
  Table T2 is a parent of table T3; delete rule CASCADE.

  CREATE VIEW V1 AS SELECT * FROM T1 UNION ALL SELECT * FROM T2

  DELETE FROM V1
  ```

  If table T1 is a parent of table T3 with a delete rule of RESTRICT, a restrict violation will be raised (SQLSTATE 23001) if there are any child rows for parent keys of T1 in T3.

  If table T1 is a parent of table T3 with a delete rule of NO ACTION, the child rows may be deleted by the delete rule of CASCADE when deleting rows from T2 before the NO ACTION delete rule is enforced for the deletes from T1. If

deletes from T2 did not result in deleting all child rows for parent keys of T1 in T3, then a constraint violation will be raised (SQLSTATE 23504).

Note that the SQLSTATE returned is different depending on whether the delete or update rule is RESTRICT or NO ACTION.

- For tables in table spaces defined on multiple partition database partition groups, table collocation should be considered when choosing the distribution keys. Following is a list of items to consider:
  - The tables must be in the same database partition group for collocation. The table spaces may be different, but must be defined in the same database partition group.
  - The distribution keys of the tables must have the same number of columns, and the corresponding key columns must be database partition-compatible for collocation.
  - The choice of distribution key also has an impact on performance of joins. If a table is frequently joined with another table, you should consider the joining column(s) as a distribution key for both tables.
- The NOT LOGGED INITIALLY option is useful for situations where a large result set needs to be created with data from an alternate source (another table or a file) and recovery of the table is not necessary. Using this option will save the overhead of logging the data. The following considerations apply when this option is specified:
  - When the unit of work is committed, all changes that were made to the table during the unit of work are flushed to disk.
  - When you run the rollforward utility and it encounters a log record that indicates that a table in the database was either populated by the Load utility or created with the NOT LOGGED INITIALLY option, the table will be marked as unavailable. The table will be dropped by the rollforward utility if it later encounters a DROP TABLE log. Otherwise, after the database is recovered, an error will be issued if any attempt is made to access the table (SQLSTATE 55019). The only operation permitted is to drop the table.
  - Once such a table is backed up as part of a database or table space back up, recovery of the table becomes possible.
- A REFRESH DEFERRED system-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes system-maintained materialized query tables. A REFRESH DEFERRED user-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes user-maintained materialized query tables. A REFRESH IMMEDIATE materialized query table defined with ENABLE QUERY OPTIMIZATION is always considered for optimization. For this optimization to be able to use a REFRESH DEFERRED or a REFRESH IMMEDIATE materialized query table, the fullselect must conform to certain rules in addition to those already described:
  - The fullselect must be a subselect with a GROUP BY clause or a subselect with a single table reference.
  - The fullselect must not include any special registers and built-in functions that depend on the value of a special register.
  - The fullselect must not include any global variables.
  - The fullselect must not include functions that are not deterministic.

  - The fullselect must not include a FETCH FIRST *n* ROWS ONLY clause.
  - The fullselect must not include an ORDER BY clause.

  If the query specified when creating a materialized query table does not conform to these rules, a warning is returned (SQLSTATE 01633).
- If a materialized query table is defined with REFRESH IMMEDIATE, or a staging table is defined with PROPAGATE IMMEDIATE, it is possible for an error to occur when attempting to apply the change resulting from an insert, update, or delete operation on an underlying table. The error will cause the failure of the insert, update, or delete operation on the underlying table.
- Materialized query tables or staging tables cannot be used as exception tables when constraints are checked in bulk, such as during load operations or during execution of the SET INTEGRITY statement.
- Certain operations cannot be performed on a table that is referenced by a materialized query table defined with REFRESH IMMEDIATE, or defined with REFRESH DEFERRED with an associated staging table:
  - IMPORT REPLACE cannot be used.
  - ALTER TABLE NOT LOGGED INITIALLY WITH EMPTY TABLE cannot be done.
- In a federated system, nicknames for relational data sources or local tables can be used as the underlying tables to create a materialized query table. Nicknames for non-relational data sources are not supported. When a nickname is one of the underlying tables, the REFRESH DEFERRED option must be used. System-maintained materialized query tables that reference nicknames are not supported in a partitioned database environment.
- ***Considerations for transaction-start-ID columns:*** A transaction-start-ID column contains a null value if the column allows null values, and there is a row-begin column and the value of the column is unique from values for row-begin columns generated for other transactions. Given that the column may contain null values, it is recommended that one of the following methods be used when retrieving a value from the column:

  ```
  COALESCE ( transaction_start_id_col,  row_begin_col)

  CASE WHEN transaction_start_id_col IS NOT NULL
            THEN transaction_start_id_col
       ELSE row_begin_col END
  ```
- ***Defining a system-period temporal table:*** A system-period temporal table definition includes the following:
  - A system period named SYSTEM_TIME, which is defined using a row-begin column and a row-end column. See the descriptions of AS ROW BEGIN, AS ROW END, and period-definition.
  - A transaction-start-ID column. See the description of AS TRANSACTION START ID.
  - A system-period data versioning definition specified on a subsequent ALTER TABLE statement that specifies the ADD VERSIONING action, which includes the name of the associated history table. See the description of the ADD VERSIONING clause under ALTER TABLE.

  To ensure that the history table cannot be implicitly dropped when a system-period temporal table is dropped, use the WITH RESTRICT ON DROP clause in the definition of the history table. A history table can manually be dropped only when the RESTRICT ON DROP attribute is removed by an ALTER TABLE statement.

- ***Defining an application-period temporal table:*** An application-period temporal table definition includes an application period named BUSINESS_TIME. The application period is defined using a begin timestamp column and an end column. See the description of period-definition.

  Data change operations on an application-period temporal table may result in an automatic insert of one or two additional rows when a row is updated or deleted. When an update or delete of a row in an application-period temporal table is specified for a portion of the period represented by that row, the row is updated or deleted and one or two rows are automatically inserted to represent the portion of the row that is not changed. New values are generated for each generated column in an application-period temporal table for each row that is automatically inserted as a result of an update or delete operation on the table. If a generated column is defined as part of a unique or primary key, parent key in a referential constraint, or unique index, it is possible that an automatic insert will violate a constraint or index in which case an error is returned.
- ***Considerations for implicitly hidden columns:*** Creating a table with implicitly hidden columns can impact the behavior of data movement utilities that are working with the table. When a table contains implicitly hidden columns, utilities like IMPORT, INGEST, and LOAD require that you specify whether data for the hidden columns is included in the operation. For example, this might mean that a load operation runs successfully against a table without any hidden columns, but fails when run against a table that contains implicitly hidden columns (SQLCODE SQL2437N). Similarly, EXPORT requires that you specify whether data for the hidden columns is included in the operation.

  Data movement utilities must use the DB2_DMU_DEFAULT registry variable, or the **implicitlyhiddeninclude** or **implicitlyhiddenmissing** file type modifiers when working with tables that contain implicitly hidden columns.
- ***Transparent DDL:*** In a federated system, a remote base table can be created, altered, or dropped using DB2 SQL. This capability is known as *transparent DDL*. Before a remote base table can be created on a data source, the federated server must be configured to access that data source. This configuration includes creating the wrapper for the data source, supplying the server definition for the server where the remote base table will be located, and creating the user mappings between the federated server and the data source.

  Transparent DDL does impose some limitations on what can be included in the CREATE TABLE statement:
  - Only columns and a primary key can be created on the remote base table.
  - Specific clauses supported by transparent DDL include:
    - *column-definition* and *unique-constraint* in the *element-list* clause
    - NOT NULL and PRIMARY KEY in the *column-options* clause
    - OPTIONS
  - The remote data source must support:
    - The remote column data types to which the DB2 column data types are mapped
    - The primary key option in the CREATE TABLE statement

    Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.

  When a remote base table is created using transparent DDL, a nickname is automatically created for that remote base table.
- A referential constraint may be defined in such a way that either the parent table or the dependent table is a part of a table hierarchy. In such a case, the effect of the referential constraint is as follows:

1. Effects of INSERT, UPDATE, and DELETE statements:
   - If a referential constraint exists, in which PT is a parent table and DT is a dependent table, the constraint ensures that for each row of DT (or any of its subtables) that has a non-null foreign key, a row exists in PT (or one of its subtables) with a matching parent key. This rule is enforced against any action that affects a row of PT or DT, regardless of how that action is initiated.
2. Effects of DROP TABLE statements:
   - for referential constraints in which the dropped table is the parent table or dependent table, the constraint is dropped
   - for referential constraints in which a supertable of the dropped table is the parent table the rows of the dropped table are considered to be deleted from the supertable. The referential constraint is checked and its delete rule is invoked for each of the deleted rows.
   - for referential constraints in which a supertable of the dropped table is the dependent table, the constraint is not checked. Deletion of a row from a dependent table cannot result in violation of a referential constraint.

- ***Privileges:*** When any table is created, the definer of the table is granted CONTROL privilege. When a subtable is created, the SELECT privilege that each user or group has on the immediate supertable is automatically granted on the subtable with the table definer as the grantor.
- ***Row size limit:*** The maximum number of bytes allowed in the row of a table is dependent on the page size of the table space in which the table is created (*tablspace-name1*). The following list shows the row size limit and number of columns limit associated with each table space page size.

Table 21. Limits for Number of Columns and Row Size in Each Table Space Page Size

Page Size	Row Size Limit	Column Count Limit
4K	4 005	500
8K	8 101	1 012
16K	16 293	1 012
32K	32 677	1 012

The actual number of columns for a table can be further limited by the following formula:

```
Total Columns * 8 + Number of LOB Columns * 12 <=
  Row Size Limit for Page Size
```

- ***Byte counts:*** The following table contains the byte counts of columns by data type. This is used to calculate the row size. The byte counts depend on whether or not VALUE COMPRESSION is active. When VALUE COMPRESSION is not active, the byte counts also depend on whether or not the column is nullable. The byte counts shown apply when row compression is not enabled. If row compression is active, the total number of bytes used by a row will generally be smaller than for an uncompressed version of the row; it will never be larger.

  If a table is based on a structured type, an additional 4 bytes of overhead is reserved to identify rows of subtables, regardless of whether or not subtables are defined. Additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable.

Table 22. Byte Counts of Columns by Data Type

Data type	VALUE COMPRESSION is active[1]	VALUE COMPRESSION is not active	
		Column is nullable	Column is not nullable
SMALLINT	4	3	2
INTEGER	6	5	4
BIGINT	10	9	8
REAL	6	5	4
DOUBLE	10	9	8
DECIMAL	The integral part of (p/2)+3, where p is the precision	The integral part of (p/2)+2, where p is the precision	The integral part of (p/2)+1, where p is the precision
DECFLOAT(16)	10	9	8
DECFLOAT(34)	18	17	16
CHAR(n)	n+2	n+1	n
VARCHAR(n)	n+2	n+5 (within a table)	n+4 (within a table)
LONG VARCHAR[2]	22	25	24
GRAPHIC(n)	n*2+2	n*2+1	n*2
VARGRAPHIC(n)	n*2+2	n*2+5 (within a table)	n*2+4 (within a table)
LONG VARGRAPHIC[2]	22	25	24
DATE	6	5	4
TIME	5	4	3
TIMESTAMP(p)	The integral part of (p+1)/2+9, where p is the precision of fractional seconds	The integral part of (p+1)/2+8, where $p$is the precision of fractional seconds	The integral part of (p+1)/2+7, where $p$is the precision of fractional seconds
XML (without INLINE LENGTH specified)	82	85	84
XML (with INLINE LENGTH specified)	INLINE LENGTH +2	INLINE LENGTH +4	INLINE LENGTH +3
Maximum LOB[3] length 1024 (without INLINE LENGTH specified)	70	73	72
Maximum LOB length 8192 (without INLINE LENGTH specified)	94	97	96
Maximum LOB length 65 536 (without INLINE LENGTH specified)	118	121	120
Maximum LOB length 524 000 (without INLINE LENGTH specified)	142	145	144
Maximum LOB length 4 190 000 (without INLINE LENGTH specified)	166	169	168
Maximum LOB length 134 000 000 (without INLINE LENGTH specified)	198	201	200

Table 22. Byte Counts of Columns by Data Type (continued)

Data type	VALUE COMPRESSION is active[1]	VALUE COMPRESSION is not active	
		Column is nullable	Column is not nullable
Maximum LOB length 536 000 000 (without INLINE LENGTH specified)	222	225	224
Maximum LOB length 1 070 000 000 (without INLINE LENGTH specified)	254	257	256
Maximum LOB length 1 470 000 000 (without INLINE LENGTH specified)	278	281	280
Maximum LOB length 2 147 483 647 (without INLINE LENGTH specified)	314	317	316
LOB with INLINE LENGTH specified	INLINE LENGTH + 2	INLINE LENGTH + 5	INLINE LENGTH + 4

[1] There is an additional 2 bytes of storage used by each row when VALUE COMPRESSION is active for that row.

[2]The LONG VARCHAR and LONG VARGRAPHIC data types are supported but are deprecated and might be removed in a future release.

[3] Each LOB value has a *LOB descriptor* in the base record that points to the location of the actual value. The size of the descriptor varies according to the maximum length defined for the column.

For a *distinct type*, the byte count is equivalent to the length of the source type of the distinct type. For a *reference type*, the byte count is equivalent to the length of the built-in data type on which the reference type is based. For a *structured type*, the byte count is equivalent to the INLINE LENGTH + 4. The INLINE LENGTH is the value specified (or implicitly calculated) for the column in the *column-options* clause.

The row sizes for the following sample tables assume that VALUE COMPRESSION is not specified:

```
DEPARTMENT 63 (0 + 3 + 33 + 7 +  3 + 17)
ORG        57 (0 + 3 + 19 + 2 + 15 + 18)
```

If VALUE COMPRESSION were to be specified, the row sizes would change to:

```
DEPARTMENT 69 (2 + 5 + 31 + 8 +  5 + 18)
ORG        53 (2 + 4 + 16 + 4 + 12 + 15)
```

- ***Storage byte counts:*** The following tables describe the storage byte counts of columns by data type for data values.

  The first table defines the sets of attributes. Those attributes are referenced in the second table, which contains the details for the byte counts for each data type.

  The byte counts depend on whether VALUE COMPRESSION is active. When VALUE COMPRESSION is not active, the byte counts also depend on whether the column is nullable. The values in the table represent the amount of storage (in bytes) that is used to store the value. The byte counts shown apply when row compression is not enabled. If row compression is active, the total number of bytes used by a row will generally be smaller than for an uncompressed version of the row; it will never be larger.

Table 23. Definitions of the criteria referenced in the related table

Case	Data value	VALUE COMPRESSION	Column nullability
A	NULL	Not active	Nullable

Table 23. Definitions of the criteria referenced in the related table (continued)

Case	Data value	VALUE COMPRESSION	Column nullability
B	NULL	Active [2]	Nullable
C	Zero-length	Active [2]	Not applicable
D	System default[1]	Active [2]	Not applicable
E	All other data values	Not active	Nullable
F	All other data values	Not active	Not nullable
G	All other data values	Active [2]	Not applicable

[1] When COMPRESS SYSTEM DEFAULT is specified for the column.

[2] There is an additional 2 bytes of storage used by each row when VALUE COMPRESSION is active for that row.

Table 24. Storage Byte Counts Based on Row Format, Data Type, and Data Value

Data type	Case A	Case B	Case C	Case D	Case E	Case F	Case G
SMALLINT	3	3	-	3	3	2	4
INTEGER	5	3	-	3	5	4	6
BIGINT	9	3	-	3	9	8	10
REAL	5	3	-	3	5	4	6
DOUBLE	9	3	-	3	9	8	10
DECIMAL	The integral part of (*p*/2)+2, where *p* is the precision	3	-	3	The integral part of (*p*/2)+2, where *p* is the precision	The integral part of (*p*/2)+1, where *p* is the precision	The integral part of (*p*/2)+3, where *p* is the precision
DECFLOAT(16)	9	3	-	3	9	8	10
DECFLOAT(34)	17	3	-	3	17	16	18
CHAR(*n*)	*n*+1	3	-	3	*n*+1	*n*	*n*+2
VARCHAR(*n*)	5	3	2	2	*N*+5, where *N* is the number of bytes in the data	*N*+4, where *N* is the number of bytes in the data	*N*+2, where *N* is the number of bytes in the data
LONG VARCHAR[2]	5	3	2	2	25	24	22
GRAPHIC(*n*)	*n**2+1	3	-	3	*n**2+1	*n**2	*n**2+2
VARGRAPHIC(*n*)	5	3	2	2	*N**2+5, where *N* is the number of bytes in the data	*N**2+4, where *N* is the number of bytes in the data	*N**2+2, where *N* is the number of bytes in the data
LONG VARGRAPHIC[2]	5	3	2	2	25	24	22
DATE	5	3	-	-	5	4	6
TIME	4	3	-	-	4	3	5
TIMESTAMP(*p*)	The integral part of (*p*+1)/2+8, where *p* is the precision of fractional seconds	3	-	-	The integral part of (*p*+1)/2+8, where *p* is the precision of fractional seconds	The integral part of (*p*+1)/2+7, where *p* is the precision of fractional seconds	The integral part of (*p*+1)/2+9, where *p* is the precision of fractional seconds
Maximum LOB[1] length 1024	5	3	2	2	(60 to 68)+5	(60 to 68)+4	(60 to 68)+2

Table 24. Storage Byte Counts Based on Row Format, Data Type, and Data Value (continued)

Data type	Case A	Case B	Case C	Case D	Case E	Case F	Case G
Maximum LOB length 8192	5	3	2	2	(60 to 92)+5	(60 to 92)+4	(60 to 92)+2
Maximum LOB length 65 536	5	3	2	2	(60 to 116)+5	(60 to 116)+4	(60 to 116)+2
Maximum LOB length 524 000	5	3	2	2	(60 to 140)+5	(60 to 140)+4	(60 to 140)+2
Maximum LOB length 4 190 000	5	3	2	2	(60 to 164)+5	(60 to 164)+4	(60 to 164)+2
Maximum LOB length 134 000 000	5	3	2	2	(60 to 196)+5	(60 to 196)+4	(60 to 196)+2
Maximum LOB length 536 000 000	5	3	2	2	(60 to 220)+5	(60 to 220)+4	(60 to 220)+2
Maximum LOB length 1 070 000 000	5	3	2	2	(60 to 252)+5	(60 to 252)+4	(60 to 252)+2
Maximum LOB length 1 470 000 000	5	3	2	2	(60 to 276)+5	(60 to 276)+4	(60 to 276)+2
Maximum LOB length 2 147 483 647	5	3	2	2	(60 to 312)+5	(60 to 312)+4	(60 to 312)+2
XML	5	3	-	-	85	84	82

[1] When COMPRESS SYSTEM DEFAULT is specified for the column.

[2] The LONG VARCHAR and LONG VARGRAPHIC data types are supported but are deprecated and might be removed in a future release.

- ***Dimension columns:*** Because each distinct value of a dimension column is assigned to a different block of the table, clustering on an expression may be desirable, such as "INTEGER(ORDER_DATE)/100". In this case, a generated column can be defined for the table, and this generated column may then be used in the ORGANIZE BY DIMENSIONS clause. If the expression is monotonic with respect to a column of the table, DB2 may use the dimension index to satisfy range predicates on that column. For example, if the expression is simply *column-name + some-positive-constant*, it is monotonic increasing. User-defined functions, certain built-in functions, and using more than one column in an expression, prevent monotonicity or its detection.

  Dimensions involving generated columns whose expressions are non-monotonic, or whose monotonicity cannot be determined, can still be created, but range queries along slice or cell boundaries of these dimensions are not supported. Equality and IN predicates *can* be processed by slices or cells.

  A generated column is monotonic if the following is true with respect to the generating function, fn:

  – Monotonic increasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)>fn(x1). For example:

    ```
    SALARY - 10000
    ```

  – Monotonic decreasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)<fn(x1). For example:

    ```
    -SALARY
    ```

  - Monotonic non-decreasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)>=fn(x1). For example:

    ```
    SALARY/1000
    ```

  - Monotonic non-increasing.

    For every possible pair of values x1 and x2, if x2>x1, then fn(x2)<=fn(x1). For example:

    ```
    -SALARY/1000
    ```

  The expression "PRICE*DISCOUNT" is not monotonic, because it involves more than one column of the table.

- ***Range-clustered tables:*** Organizing a table by key sequence is effective for certain types of tables. The table should have an integer key that is tightly clustered (dense) over the range of possible values. The columns of this integer key must not be nullable, and the key should logically be the primary key of the table. The organization of a range-clustered table precludes the need for a separate unique index object, providing direct access to the row for a specified key value, or a range of rows for a specified range of key values. The allocation of all the space for the complete set of rows in the defined key sequence range is done during table creation, and must be considered when defining a range-clustered table. The storage space is not available for any other use, even though the rows are initially marked deleted. If the full key sequence range will be populated with data only over a long period of time, this table organization may not be an appropriate choice.
- A table can have at most one security policy.
- DB2 enforces referential integrity constraints that are defined on protected tables. Constraints violations in this case can be difficult to debug, because DB2 will not allow you to see what row has caused a violation if you do not have the appropriate security label or exemptions credentials.
- When defining the order of columns in a table, frequently updated columns should be placed at the end of the definition to minimize the amount of data logged for updates. This includes ROW CHANGE TIMESTAMP columns. ROW CHANGE TIMESTAMP columns are guaranteed to be updated on each row update.
- ***Security and replication:*** Replication can cause data rows from a protected table to be replicated outside of the database. Care must be taken when setting up replication for a protected table, because DB2 cannot protect data that is outside of the database.
- ***Considerations for a multi-partition or DB2 pureScale environment***:
  - In a multi-partition or DB2 pureScale environment, if the CACHE and NO ORDER options are in effect, multiple caches can be active simultaneously and the requests for next value assignments from different members might not result in the assignment of values in strict numeric order. Assume, for example, that members DB2A and DB2B are using the same sequence, and DB2A gets the cache values 1 to 20 and DB2B gets the cache values 21 to 40. In this scenario, if DB2A requested the next value first, then DB2B requested, and then DB2A requested again, the actual order of values assigned would be 1,21,2. Therefore, to guarantee that sequence numbers are generated in strict numeric order among multiple members using the same sequence concurrently, specify the ORDER option.
  - In a DB2 pureScale environment, using the ORDER or NO CACHE option ensures that the values assigned to a sequence which is shared by

applications across multiple members are in strict numeric order. In a DB2 pureScale environment, if ORDER is specified, then NO CACHE is implied even if CACHE *n* is specified

- ***Considerations for row and column access control (RCAC)***: The ACTIVATE ROW ACCESS CONTROL, ACTIVATE COLUMN ACCESS CONTROL, DEACTIVATE ROW ACCESS CONTROL, and DEACTIVATE COLUMN ACCESS CONTROL clauses are not supported. Use the ALTER TABLE statement to activate or deactivate row or column level access control on a table.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - The following syntax is accepted as the default behavior:
    - IN database-name.tablespace-name
    - IN DATABASE database-name
    - FOR MIXED DATA
    - FOR SBCS DATA
  - PART can be specified in place of PARTITION
  - PARTITION *partition-number* can be specified instead of PARTITION *partition-name*. A *partition-number* must not identify a partition that was previously specified in the CREATE TABLE statement. If a *partition-number* is not specified, a unique partition number is generated by the database manager.
  - VALUES can be specified in place of ENDING AT
  - The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause
  - *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword)
  - SUMMARY can optionally be specified after CREATE
  - DEFINITION ONLY can be specified in place of WITH NO DATA
  - The PARTITIONING KEY clause can be specified in place of the DISTRIBUTE BY clause
  - REPLICATED can be specified in place of DISTRIBUTE BY REPLICATION
  - A comma can be used to separate multiple options in the *identity-options* clause
  - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER can be specified in place of NO MINVALUE, NO MAXVALUE, NO CYCLE, NO CACHE, and NO ORDER, respectively
  - ADD can be specified before *table-option-name string-constant*.

Examples

- *Example 1:* Create table TDEPT in the DEPARTX table space. DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT are column names. CHAR means the column will contain character data. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. The primary key consists of the column DEPTNO.

```
CREATE TABLE TDEPT
  (DEPTNO   CHAR(3)     NOT NULL,
   DEPTNAME VARCHAR(36) NOT NULL,
   MGRNO    CHAR(6),
   ADMRDEPT CHAR(3)     NOT NULL,
   PRIMARY KEY(DEPTNO))
IN DEPARTX
```

- *Example 2:* Create table PROJ in the SCHED table space. PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF, PRSTDATE, PRENDATE, and MAJPROJ are column names. CHAR means the column will contain character data. DECIMAL means the column will contain packed decimal data. 5,2 means the following: 5 indicates the number of decimal digits, and 2 indicates the number of digits to the right of the decimal point. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. DATE means the column will contain date information in a three-part format (year, month, and day).

```
CREATE TABLE PROJ
  (PROJNO   CHAR(6)      NOT NULL,
   PROJNAME VARCHAR(24)  NOT NULL,
   DEPTNO   CHAR(3)      NOT NULL,
   RESPEMP  CHAR(6)      NOT NULL,
   PRSTAFF  DECIMAL(5,2)         ,
   PRSTDATE DATE                 ,
   PRENDATE DATE                 ,
   MAJPROJ  CHAR(6)      NOT NULL)
IN SCHED
```

- *Example 3:* Create a table called EMPLOYEE_SALARY where any unknown salary is considered 0. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the *IN tablespace-name* clause.

```
CREATE TABLE EMPLOYEE_SALARY
  (DEPTNO   CHAR(3)      NOT NULL,
   DEPTNAME VARCHAR(36)  NOT NULL,
   EMPNO    CHAR(6)      NOT NULL,
   SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT)
```

- *Example 4:* Create distinct types for total salary and miles and use them for columns of a table created in the default table space. In a dynamic SQL statement assume the CURRENT SCHEMA special register is JOHNDOE and the CURRENT PATH is the default ("SYSIBM","SYSFUN","JOHNDOE").

  If a value for SALARY is not specified it must be set to 0 and if a value for LIVING_DIST is not specified it must to set to 1 mile.

```
CREATE TYPE JOHNDOE.T_SALARY AS INTEGER WITH COMPARISONS

CREATE TYPE JOHNDOE.MILES  AS FLOAT WITH COMPARISONS

CREATE TABLE EMPLOYEE
  (ID          INTEGER NOT NULL,
   NAME        CHAR (30),
   SALARY      T_SALARY NOT NULL WITH DEFAULT,
   LIVING_DIST MILES    DEFAULT MILES(1) )
```

- *Example 5:* Create distinct types for image and audio and use them for columns of a table. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the IN *tablespace-name* clause. Assume the CURRENT PATH is the default.

```
CREATE TYPE IMAGE AS BLOB (10M)

CREATE TYPE AUDIO AS BLOB (1G)

CREATE TABLE PERSON
  (SSN    INTEGER NOT NULL,
   NAME   CHAR (30),
   VOICE  AUDIO,
   PHOTO  IMAGE)
```

- *Example 6:* Create table EMPLOYEE in the HUMRES table space. The constraints defined on the table are the following:
  - The values of department number must lie in the range 10 to 100.

- The job of an employee can only be either 'Sales', 'Mgr' or 'Clerk'.
- Every employee that has been with the company since 1986 must make more than $40,500.

Note: If the columns included in the check constraints are nullable they could also be NULL.

```
CREATE TABLE EMPLOYEE
  (ID          SMALLINT NOT NULL,
   NAME        VARCHAR(9),
   DEPT        SMALLINT CHECK (DEPT BETWEEN 10 AND 100),
   JOB         CHAR(5) CHECK (JOB IN ('Sales','Mgr','Clerk')),
   HIREDATE    DATE,
   SALARY      DECIMAL(7,2),
   COMM        DECIMAL(7,2),
   PRIMARY KEY (ID),
   CONSTRAINT  YEARSAL CHECK (YEAR(HIREDATE) > 1986
     OR SALARY > 40500)
  )
  IN HUMRES
```

- *Example 7:* Create a table that is wholly contained in the PAYROLL table space.

```
CREATE TABLE EMPLOYEE .....
  IN PAYROLL
```

- *Example 8:* Create a table with its data part in ACCOUNTING and its index part in ACCOUNT_IDX.

```
CREATE TABLE SALARY.....
  IN ACCOUNTING INDEX IN ACCOUNT_IDX
```

- *Example 9:* Create a table and log SQL changes in the default format.

```
CREATE TABLE SALARY1 .....
```

or

```
CREATE TABLE SALARY1 .....
  DATA CAPTURE NONE
```

- *Example 10:* Create a table and log SQL changes in an expanded format.

```
CREATE TABLE SALARY2 .....
  DATA CAPTURE CHANGES
```

- *Example 11:* Create a table EMP_ACT in the SCHED table space. EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, and EMENDATE are column names. Constraints defined on the table are:
  - The value for the set of columns, EMPNO, PROJNO, and ACTNO, in any row must be unique.
  - The value of PROJNO must match an existing value for the PROJNO column in the PROJECT table and if the project is deleted all rows referring to the project in EMP_ACT should also be deleted.

```
CREATE TABLE EMP_ACT
  (EMPNO       CHAR(6) NOT NULL,
   PROJNO      CHAR(6) NOT NULL,
   ACTNO       SMALLINT NOT NULL,
   EMPTIME     DECIMAL(5,2),
   EMSTDATE    DATE,
   EMENDATE    DATE,
   CONSTRAINT EMP_ACT_UNIQ UNIQUE (EMPNO,PROJNO,ACTNO),
   CONSTRAINT FK_ACT_PROJ FOREIGN KEY (PROJNO)
                          REFERENCES PROJECT (PROJNO) ON DELETE CASCADE
  )
  IN SCHED
```

A unique index called EMP_ACT_UNIQ is automatically created in the same schema to enforce the unique constraint.

- *Example 12:* Create a table that is to hold information about famous goals for the ice hockey hall of fame. The table will list information about the player who scored the goal, the goaltender against who it was scored, the date, and a description. The description column is nullable.

```
CREATE TABLE HOCKEY_GOALS
  ( BY_PLAYER      VARCHAR(30)   NOT NULL,
    BY_TEAM        VARCHAR(30)   NOT NULL,
    AGAINST_PLAYER VARCHAR(30)   NOT NULL,
    AGAINST_TEAM   VARCHAR(30)   NOT NULL,
    DATE_OF_GOAL   DATE          NOT NULL,
    DESCRIPTION    CLOB(5000) )
```

- *Example 13:* Suppose an exception table is needed for the EMPLOYEE table. One can be created using the following statement.

```
CREATE TABLE EXCEPTION_EMPLOYEE AS
  (SELECT EMPLOYEE.*,
    CURRENT TIMESTAMP AS TIMESTAMP,
    CAST ('' AS CLOB(32K)) AS MSG
  FROM EMPLOYEE
  ) WITH NO DATA
```

- *Example 14:* Given the following table spaces with the indicated attributes:

```
TBSPACE            PAGESIZE    USER   USERAUTH
------------------ ----------- ------ --------
DEPT4K                    4096 BOBBY  Y
PUBLIC4K                  4096 PUBLIC Y
DEPT8K                    8192 BOBBY  Y
DEPT8K                    8192 RICK   Y
PUBLIC8K                  8192 PUBLIC Y
```

  – If RICK creates the following table, it is placed in table space PUBLIC4K since the byte count is less than 4005; but if BOBBY creates the same table, it is placed in table space DEPT4K, since BOBBY has USE privilege because of an explicit grant:

```
CREATE TABLE DOCUMENTS
  (SUMMARY    VARCHAR(1000),
   REPORT     VARCHAR(2000))
```

  – If BOBBY creates the following table, it is placed in table space DEPT8K since the byte count is greater than 4005, and BOBBY has USE privilege because of an explicit grant. However, if DUNCAN creates the same table, it is placed in table space PUBLIC8K, since DUNCAN has no specific privileges:

```
CREATE TABLE CURRICULUM
  (SUMMARY    VARCHAR(1000),
   REPORT     VARCHAR(2000),
   EXERCISES  VARCHAR(1500))
```

- *Example 15:* Create a table with a LEAD column defined with the structured type EMP. Specify an INLINE LENGTH of 300 bytes for the LEAD column, indicating that any instances of LEAD that cannot fit within the 300 bytes are stored outside the table (separately from the base table row, similar to the way LOB values are handled).

```
CREATE TABLE PROJECTS (PID INTEGER,
  LEAD EMP INLINE LENGTH 300,
  STARTDATE DATE,
            ...)
```

- *Example 16:* Create a table DEPT with five columns named DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, and LOCATION. Column DEPT is to be defined as an IDENTITY column such that DB2 will always generate a value for it. The values for the DEPT column should begin with 500 and increment by 1.

```
CREATE TABLE DEPT
  (DEPTNO     SMALLINT      NOT NULL
                GENERATED ALWAYS AS IDENTITY
                  (START WITH 500, INCREMENT BY 1),
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30))
```

- *Example 17:* Create a SALES table that is distributed on the YEAR column, and that has dimensions on the REGION and YEAR columns. Data will be distributed across database partitions according to hashed values of the YEAR column. On each database partition, data will be organized into extents based on unique combinations of values of the REGION and YEAR columns on those database partitions.

```
CREATE TABLE SALES
  (CUSTOMER   VARCHAR(80),
   REGION     CHAR(5),
   YEAR       INTEGER)
DISTRIBUTE BY HASH (YEAR)
ORGANIZE BY DIMENSIONS (REGION, YEAR)
```

- *Example 18:* Create a SALES table with a PURCHASEYEARMONTH column that is generated from the PURCHASEDATE column. Use an expression to create a column that is monotonic with respect to the original PURCHASEDATE column, and is therefore suitable for use as a dimension. The table is distributed on the REGION column, and organized within each database partition into extents according to the PURCHASEYEARMONTH column; that is, different regions will be on different database partitions, and different purchase months will belong to different cells (or sets of extents) within those database partitions.

```
CREATE TABLE SALES
  (CUSTOMER            VARCHAR(80),
   REGION              CHAR(5),
   PURCHASEDATE        DATE,
   PURCHASEYEARMONTH   INTEGER
                         GENERATED ALWAYS AS (INTEGER(PURCHASEDATE)/100))
DISTRIBUTE BY HASH (REGION)
ORGANIZE BY DIMENSIONS (PURCHASEYEARMONTH)
```

- *Example 19:* Create a CUSTOMER table with a CUSTOMERNUMDIM column that is generated from the CUSTOMERNUM column. Use an expression to create a column that is monotonic with respect to the original CUSTOMERNUM column, and is therefore suitable for use as a dimension. The table is organized into cells according to the CUSTOMERNUMDIM column, so that there is a different cell in the table for every 50 customers. If a unique index were created on CUSTOMERNUM, customer numbers would be clustered in such a way that each set of 50 values would be found in a particular set of extents in the table.

```
CREATE TABLE CUSTOMER
  (CUSTOMERNUM      INTEGER,
   CUSTOMERNAME     VARCHAR(80),
   ADDRESS          VARCHAR(200),
   CITY             VARCHAR(50),
   COUNTRY          VARCHAR(50),
   CODE             VARCHAR(15),
   CUSTOMERNUMDIM   INTEGER
                      GENERATED ALWAYS AS (CUSTOMERNUM/50))
ORGANIZE BY DIMENSIONS (CUSTOMERNUMDIM)
```

- *Example 20:* Create a remote base table called EMPLOYEE on the Oracle server, ORASERVER. A nickname, named EMPLOYEE, which refers to this newly created remote base table, will also automatically be created.

```
CREATE TABLE EMPLOYEE
  (EMP_NO       CHAR(6)       NOT NULL,
   FIRST_NAME   VARCHAR(12)   NOT NULL,
   MID_INT      CHAR(1)       NOT NULL,
   LAST_NAME    VARCHAR(15)   NOT NULL,
   HIRE_DATE    DATE,
   JOB          CHAR(8),
   SALARY       DECIMAL(9,2),
   PRIMARY KEY (EMP_NO))
OPTIONS
  (REMOTE_SERVER 'ORASERVER',
   REMOTE_SCHEMA 'J15USER1',
   REMOTE_TABNAME 'EMPLOYEE')
```

The following CREATE TABLE statements show how to specify the table name, or the table name and the explicit remote base table name, to get the required case. The lowercase identifier, employee, is used to illustrate the implicit folding of identifiers.

Create a remote base table called EMPLOYEE (uppercase characters) on an Informix® server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```
CREATE TABLE employee
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER')
```

If the REMOTE_TABNAME option is not specified, and *table-name* is not delimited, the remote base table name will be in uppercase characters, even if the remote data source normally stores names in lowercase characters.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```
CREATE TABLE employee
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER',
   REMOTE_TABNAME 'employee')
```

When creating a table at a remote data source that supports delimited identifiers, use the REMOTE_TABNAME option and a character string constant that specifies the table name in the required case.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named employee (lowercase characters) on that table:

```
CREATE TABLE "employee"
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER')
```

If the REMOTE_TABNAME option is not specified, and *table-name* is delimited, the remote base table name will be identical to *table-name*.

- *Example 21:* Create a range-clustered table that can be used to locate a student using a student ID. For each student record, include the school ID, program ID, student number, student ID, student first name, student last name, and student grade point average (GPA).

```
CREATE TABLE STUDENTS
  (SCHOOL_ID     INTEGER   NOT NULL,
   PROGRAM_ID    INTEGER   NOT NULL,
   STUDENT_NUM   INTEGER   NOT NULL,
   STUDENT_ID    INTEGER   NOT NULL,
   FIRST_NAME    CHAR(30),
   LAST_NAME     CHAR(30),
   GPA           DOUBLE)
ORGANIZE BY KEY SEQUENCE
  (STUDENT_ID
    STARTING FROM 1
    ENDING AT 1000000)
  DISALLOW OVERFLOW
```

The size of each record is the sum of the columns, plus alignment, plus the range-clustered table row header. In this case, the row size is 98 bytes: 4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 (for nullable columns) + 1 (for alignment) + 10 (for the header). With a 4-KB page size (or 4096 bytes), after accounting for page overhead, there are 4038 bytes available, enough room for 41 records per page. Allowing for 1 million student records, there is a need for (1 million divided by 41 records per page) 24 391 pages. With two additional pages for table overhead, the final number of 4-KB pages that are allocated when the table is created is 24 393.

- *Example 22:* Create a table named DEPARTMENT with a functional dependency that has no specified constraint name.

```
CREATE TABLE DEPARTMENT
  (DEPTNO     SMALLINT      NOT NULL,
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30),
  CHECK (DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED)
```

- *Example 23:* Create a table with protected rows.

```
CREATE TABLE TOASTMASTERS
  (PERFORMANCE DB2SECURITYLABEL,
   POINTS      INTEGER,
   NAME        VARCHAR(50))
  SECURITY POLICY CONTRIBUTIONS
```

- *Example 24:* Create a table with protected columns.

```
CREATE TABLE TOASTMASTERS
  (PERFORMANCE CHAR(8),
   POINTS      INTEGER COLUMN SECURED WITH CLUBPOSITION,
   NAME        VARCHAR(50))
  SECURITY POLICY CONTRIBUTIONS
```

- *Example 25:* Create a table with protected rows and columns.

```
CREATE TABLE TOASTMASTERS
  (PERFORMANCE DB2SECURITYLABEL,
   POINTS      INTEGER COLUMN SECURED WITH CLUBPOSITION,
   NAME        VARCHAR(50))
  SECURITY POLICY CONTRIBUTIONS
```

- *Example 26:* Large objects for a partitioned table reside, by default, in the same table space as the data. This default behavior can be overridden by using the LONG IN clause to specify one or more table spaces for the large objects. Create a table named DOCUMENTS whose large object data is to be stored (in a round-robin fashion for each data partition) in table spaces TBSP1 and TBSP2.

```
CREATE TABLE DOCUMENTS
  (ID INTEGER,
   CONTENTS CLOB)
```

```
  LONG IN TBSP1, TBSP2
  PARTITION BY RANGE (ID)
    (STARTING 1 ENDING 1000
     EVERY 100)
```

Alternatively, use the long form of the syntax to explicitly identify a large table space for each data partition. In this example, the CLOB data for the first data partition is placed in LARGE_TBSP3, and the CLOB data for the remaining data partitions is spread across LARGE_TBSP1 and LARGE_TBSP2 in a round-robin fashion.

```
CREATE TABLE DOCUMENTS
  (ID INTEGER,
   CONTENTS CLOB)
  LONG IN LARGE_TBSP1, LARGE_TBSP2
  PARTITION BY RANGE (ID)
    (STARTING 1 ENDING 100
     IN TBSP1 LONG IN LARGE_TBSP3,
     STARTING 101 ENDING 1000
     EVERY 100)
```

- *Example 27:* Create a partitioned table named ACCESSNUMBERS having two data partitions. The row (10, NULL) is to be placed in the first partition, and the row (NULL, 100) is to be placed in the second (last) data partition.

  ```
  CREATE TABLE ACCESSNUMBERS
    (AREA INTEGER,
     EXCHANGE INTEGER)
    PARTITION BY RANGE (AREA NULLS LAST, EXCHANGE NULLS FIRST)
      (STARTING (1,1) ENDING (10,100),
       STARTING (11,1) ENDING (MAXVALUE,MAXVALUE))
  ```

  Because null values in the second column are sorted first, the row (11, NULL) would sort below the low boundary of the last data partition (11, 1); attempting to insert this row returns an error. The row (12, NULL) would fall within the last data partition.

- *Example 28:* Create a table named RATIO having a single data partition and partitioning column PERCENT.

  ```
  CREATE TABLE RATIO
    (PERCENT INTEGER)
    PARTITION BY RANGE (PERCENT)
      (STARTING (MINVALUE) ENDING (MAXVALUE))
  ```

  This table definition allows any integer value for column PERCENT to be inserted. The following definition for the RATIO table allows any integer value between 1 and 100 inclusive to be inserted into column PERCENT.

  ```
  CREATE TABLE RATIO
    (PERCENT INTEGER)
    PARTITION BY RANGE (PERCENT)
      (STARTING 0 EXCLUSIVE ENDING 100 INCLUSIVE)
  ```

- *Example 29:* Create a table named MYDOCS with two columns: one is an identifier, and the other stores XML documents.

  ```
  CREATE TABLE MYDOCS
    (ID INTEGER,
     DOC XML)
  IN HLTBSPACE
  ```

- *Example 30:* Create a table named NOTES with four columns, including one for storing XML-based notes.

```
CREATE TABLE NOTES
  (ID          INTEGER,
   DESCRIPTION VARCHAR(255),
   CREATED     TIMESTAMP,
   NOTE        XML)
```

- *Example 31:* Create a table, EMP_INFO, that contains a phone number and address for each employee. Include a ROW CHANGE TIMESTAMP column in the table to track the modification of employee information.

```
CREATE TABLE EMP_INFO
    (EMPNO           CHAR(6) NOT NULL,
     EMP_INFOCHANGE  TIMESTAMP NOT NULL GENERATED ALWAYS
      FOR EACH ROW ON UPDATE
      AS ROW CHANGE TIMESTAMP,
     EMP_ADDRESS     VARCHAR(300),
     EMP_PHONENO     CHAR(4),
     PRIMARY KEY (EMPNO) )
```

- *Example 32:* Create a partitioned table named DOCUMENTS having two data partitions:
  - The data object in the first partition resides in table space TBSP11. The partitioned index partition on the partition resides in table space TBSP21. The XML data object resides in table space TBSP31.
  - The data object in the second partition resides in table space TBSP12. The partitioned index partition on the partition resides in table space TBSP22. The XML data object resides in table space TBSP32.

  The table level INDEX IN clause has no impact on table space selection for partitioned indexes.

```
CREATE TABLE DOCUMENTS
(ID        INTEGER,
 CONTENTS  XML)  INDEX IN TBSPX
PARTITION BY (ID NULLS LAST)
(STARTING FROM 1 INCLUSIVE ENDING AT 100 INCLUSIVE
 IN TBSP11 INDEX IN TBSP21 LONG IN TBSP31,
 STARTING FROM 101 INCLUSIVE ENDING AT 200 INCLUSIVE
 IN TBSP21 INDEX IN TBSP22 LONG IN TBSP32)
```

- *Example 33:* Create a partitioned table named SALES having two data partitions:
  - The data object in the first partition resides in table space TBSP11. The partitioned index partition on the partition resides in table space TBSP21.
  - The data object in the second partition resides in table space TBSP12. The partitioned index object resides in table space TBSP22.

  The table level INDEX IN clause has no impact on table space selection for partitioned indexes.

```
CREATE TABLE SALES
(SID     INTEGER,
 AMOUNT  INTEGER)  INDEX IN TBSPX
PARTITION BY RANGE (SID NULLS LAST)
(STARTING FROM 1 INCLUSIVE ENDING AT 100 INCLUSIVE
 IN TBSP11 INDEX IN TBSP21,
 STARTING FROM 101 INCLUSIVE ENDING AT 200 INCLUSIVE
 IN TBSP12 INDEX IN TBSP22)
```

- *Example 34:* Create a table named BOOKS with four columns, including one named DATE_ADDED, which inserts the current TIMESTAMP by default.

```
CREATE TABLE BOOKS
  (ISBN_NUM   INTEGER,
   TITLE      VARCHAR(255),
   AUTHOR     VARCHAR(255),
   DATE_ADDED TIMESTAMP WITH DEFAULT CURRENT TIMESTAMP)
```

- *Example 35:* Create a Unicode table called STUDENTS in a non-Unicode database. Assume that the database was created using code set 1252 and territory CA and the ALT_COLLATE database configuration parameter was updated to IDENTITY_16BIT.

```
CREATE TABLE STUDENTS (
      STUDENTID INT NOT NULL,
     FAMILY_NAME VARCHAR(36) NOT NULL,
     GIVEN_NAME VARCHAR(36) NOT NULL,
     PRIMARY KEY(STUDENTID))
   CCSID UNICODE
```

- *Example 36:* Create a table called TDEPT_TEMP, based on the TDEPT table that is created in Example 1.

```
CREATE TABLE TDEPT_TEMP LIKE TDEPT
```

  The TDEPT_TEMP table will have the same definition as TDEPT except that the primary key will not be defined and a default table space will be implicitly chosen.

- *Example 37:* Column security labels inherited by a materialized query table.

```
CREATE SECURITY LABEL COMPONENT level_array ARRAY ['A', 'B', 'C']

CREATE SECURITY POLICY P COMPONENTS level_array WITH DB2LBACRULES

CREATE SECURITY LABEL P.A COMPONENT level_array 'A'

CREATE SECURITY LABEL P.B COMPONENT level_array 'B'

CREATE SECURITY LABEL P.C COMPONENT level_array 'C'

CREATE TABLE t1 (c1 INT, c2 INT SECURED WITH B, c3 REAL SECURED WITH A)
   SECURITY POLICY P

CREATE TABLE t2 (c4 REAL, c5 INT SECURED WITH C, c6 DB2SECURITYLABEL)
   SECURITY POLICY P
```

  Generate a materialized query table

```
CREATE TABLE m1 AS(SELECT  c1, c3, c5, c6 FROM  t1,t2 WHERE c2 !=100)
DATA INITALLY DEFERRED REFRESH DEFERRED
```

  The security label of t1.c2 is used to compute security labels of all columns of m1 because it appears in the predicates of the query. The label-based access control properties of the materialized query table m1 are:

  – Security policy = P
  – Security label of column m1.c1 = P.B
  – Security label of column m1.c3 = P.A
  – Security label of column m1.c5 = P.B
  – Security label of column m1.c6 = P.B and it is also DB2SECURITYLABEL.

- *Example 38:* A staging table for a materialized query table is protected with label-based access control. Following Example 33, if staging table st1 is defined as:

```
CREATE TABLE st1 FOR  m1 PROPAGATE IMMEDIATE
```

  The label-based access control properties of the staging table st1 are:

  – Security policy = P
  – Security label of column st1.c1 = P.B
  – Security label of column st1.c3 = P.A

- Security label of column st1.c5 = P.B
- Security label of column st1.c6 = P.B and it is also DB2SECURITYLABEL.

CREATE TABLESPACE

The CREATE TABLESPACE statement defines a new table space within the database, assigns containers to the table space, and records the table space definition and attributes in the catalog.

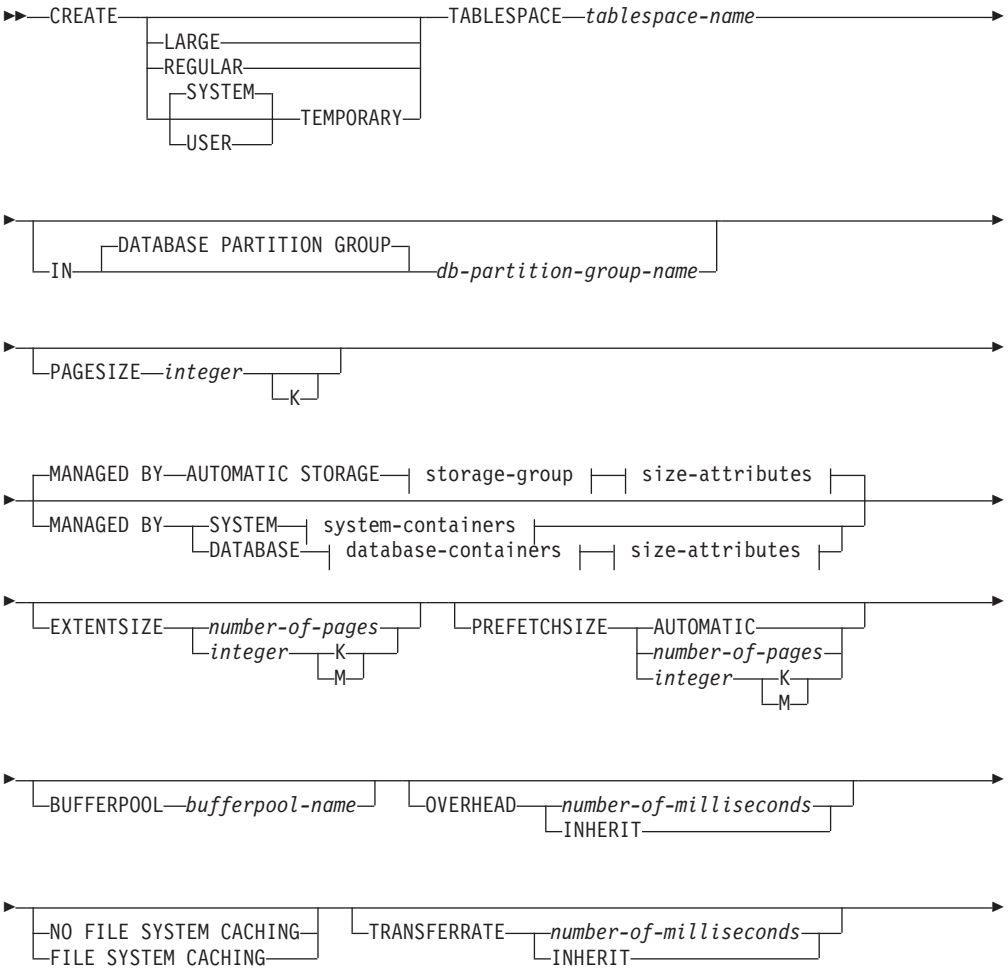
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

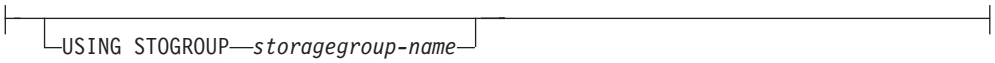
Authorization

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

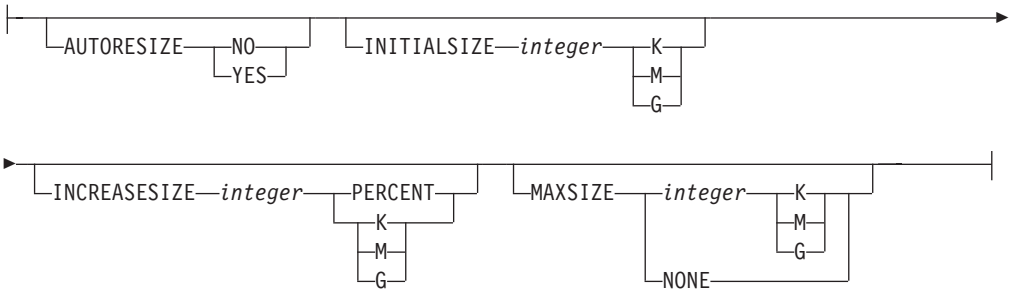
Syntax



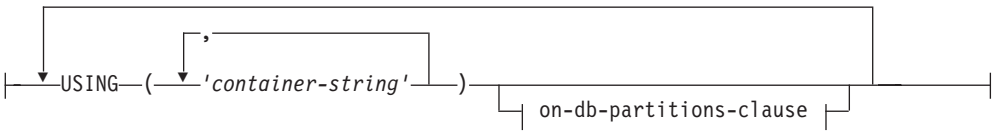
storage-group:



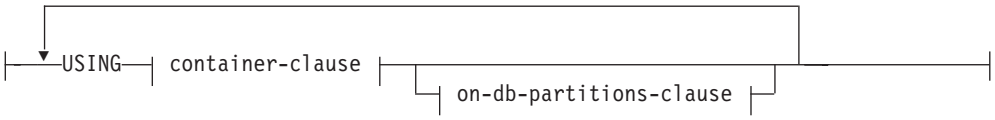
size-attributes:



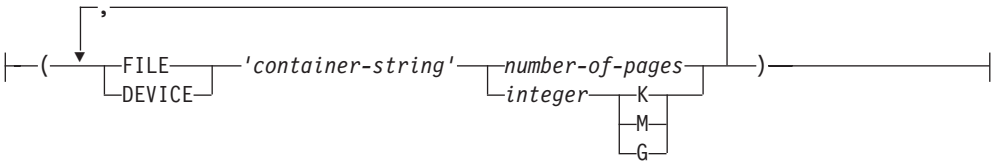
system-containers:



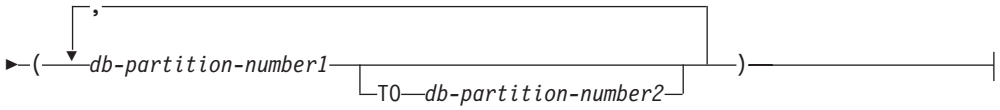
database-containers:



container-clause:



on-db-partitions-clause:



Description

LARGE, REGULAR, SYSTEM TEMPORARY, or USER TEMPORARY
Specifies the type of table space that is to be created. If no type is specified, the default is determined by the MANAGED BY clause.

LARGE
Stores all permanent data. This type is only allowed on database managed space (DMS) table spaces. It is also the default type for DMS table spaces when no type is specified. When a table is placed in a large table space:

- The table can be larger than a table in a regular table space. For details on table and table space limits, see "SQL and XML limits".
- The table can support more than 255 rows per data page, which can improve space utilization on data pages.
- Indexes that are defined on the table will require an additional 2 bytes per row entry, compared to indexes defined on a table that resides in a regular table space.

REGULAR
Stores all permanent data. This type applies to both DMS and SMS table spaces. This is the only type allowed for SMS table spaces, and it is also the default type for SMS table spaces when no type is specified.

SYSTEM TEMPORARY
Stores temporary tables, work areas used by the database manager to perform operations such as sorts or joins. A database must always have at least one SYSTEM TEMPORARY table space, because temporary tables can only be stored in such a table space. A temporary table space is created automatically when a database is created.

USER TEMPORARY
Stores created temporary tables and declared temporary tables. No user temporary table spaces exist when a database is created. To allow the definition of created temporary tables or declared temporary tables, at least one user temporary table space should be created with appropriate USE privileges.

tablespace-name
Names the table space. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *tablespace-name* must not identify a table space that already exists in the catalog (SQLSTATE 42710). The *tablespace-name* must not begin with the characters 'SYS' (SQLSTATE 42939).

IN DATABASE PARTITION GROUP *db-partition-group-name*
Specifies the database partition group for the table space. The database partition group must exist. The only database partition group that can be specified when creating a SYSTEM TEMPORARY table space is IBMTEMPGROUP. The DATABASE PARTITION GROUP keywords are optional.

If the database partition group is not specified, the default database partition group (IBMDEFAULTGROUP) is used for REGULAR, LARGE, and USER

TEMPORARY table spaces. For SYSTEM TEMPORARY table spaces, the default database partition group IBMTEMPGROUP is used.

PAGESIZE *integer* **[K]**
Defines the size of pages used for the table space. The valid values for *integer* without the suffix K are `4 096`, `8 192`, `16 384`, or `32 768`. The valid values for *integer* with the suffix K are `4`, `8`, `16`, or `32`. Any number of spaces is allowed between *integer* and K, including no space. An error occurs if the page size is not one of these values (SQLSTATE 428DE), or if the page size is not the same as the page size of the buffer pool that is associated with the table space (SQLSTATE 428CB).

The default value is provided by the **pagesize** database configuration parameter, which is set when the database is created.

MANAGED BY AUTOMATIC STORAGE
Specifies that the table space is to be an automatic storage table space. If there are no storage groups defined, an error is returned (SQLSTATE 55060).

The database manager automatically decides how the automatic storage table space is initially created. Temporary table spaces are initialized as system managed space (SMS) table space and permanent table spaces are initialized as database managed space (DMS) table space. When creating a permanent table space and the type of table space is not specified, the default behavior is to create a large table space. With an automatic storage table space, the database manager determines which containers are to be assigned to the table space, based upon the storage paths that are associated with the storage group the table space uses.

storage-group
Specify the storage group for an automatic storage table space.

USING STOGROUP
For an automatic storage table space, identifies the storage group for the table space in which the table space data will be stored. If a *storagegroup-name* is not specified, then the currently designated default storage group is used. This clause only applies to automatic storage table spaces (SQLSTATE 42613).

storagegroup-name
Identifies the storage group in which table space data will be stored. *storagegroup-name* must identify a storage group that exists at the current server (SQLSTATE 42704). This is a one-part name.

size-attributes
Specify the size attributes for an automatic storage table space or a DMS table space that is not an automatic storage table space. SMS table spaces are not auto-resizable.

AUTORESIZE
Specifies whether or not the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled. Auto-resizable table spaces automatically increase in size when they become full. The default is NO for DMS table spaces and YES for automatic storage table spaces.

NO Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be disabled.

YES
Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled.

INITIALSIZE *integer* **K | M | G**
Specifies the initial size, per database partition, of an automatic storage table space. This option is only valid for automatic storage table spaces. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain a consistent size across containers in the table space. Moreover, if the table space is auto-resizable and the initial size is not large enough to contain meta-data that must be added to the new table space, the database manager will continue to extend the table space by the value of INCREASESIZE until there is enough space. If the INITIALSIZE clause is not specified, the database manager determines an appropriate value. The value for *integer* must be at least 48 K.

INCREASESIZE *integer* **PERCENT or INCREASESIZE** *integer* **K | M | G**
Specifies the amount, per database partition, by which a table space that is enabled for auto-resize will automatically be increased when the table space is full, and a request for space has been made. The integer value must be followed by:

- PERCENT to specify the amount as a percentage of the table space size at the time that a request for space is made. When PERCENT is specified, the integer value must be between `0` and `100` (SQLSTATE 42615).
- K (for kilobytes), M (for megabytes), or G (for gigabytes) to specify the amount in bytes

Note that the actual value used might be slightly smaller or larger than what was specified, because the database manager strives to maintain consistent growth across containers in the table space. If the table space is auto-resizable, but the INCREASESIZE clause is not specified, the database manager determines an appropriate value.

MAXSIZE *integer* **K | M | G or MAXSIZE NONE**
Specifies the maximum size to which a table space that is enabled for auto-resize can automatically be increased. If the table space is auto-resizable, but the MAXSIZE clause is not specified, the default is NONE.

integer
Specifies a hard limit on the size, per database partition, to which a DMS table space or an automatic storage table space can automatically be increased. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

NONE
Specifies that the table space is to be allowed to grow to file system capacity, or to the maximum table space size (described in "SQL and XML limits").

MANAGED BY SYSTEM
Specifies that the table space is to be an SMS table space.

MANAGED BY SYSTEM cannot be specified in a DB2 pureScale environment (SQLSTATE 42997).

Important: The SMS table space type has been deprecated in Version 10.1 for user-defined permanent table spaces and might be removed in a future release. The SMS table space type is not deprecated for catalog and temporary table spaces. For more information, see "SMS permanent table spaces have been deprecated" in *What's New for DB2 Version 10.1*

system-containers
Specify the containers for an SMS table space.

USING *('container-string',...)*
For an SMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The *container-string* cannot exceed 240 bytes in length.

Each *container-string* can be an absolute or relative directory name.

The directory name, if not absolute, is relative to the database directory, and can be a path name alias (a symbolic link on UNIX systems) to storage that is not physically associated with the database directory. For example, *dbdir*/work/c1 could be a symbolic link to a separate file system.

If any component of the directory name does not exist, it is created by the database manager. When a table space is dropped, all components created by the database manager are deleted. If the directory identified by *container-string* exists, it must not contain any files or subdirectories (SQLSTATE 428B2).

The format of *container-string* is dependent on the operating system. On Windows operating systems, an absolute directory path name begins with a drive letter and a colon (:); on UNIX systems, an absolute path name begins with a forward slash (/). A relative path name on any platform does not begin with an operating system-dependent character.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only supported with the use of an uninterrupted power supply (UPS); continuous UPS (rather than standby) is recommended. An NFS-mounted file system on AIX must be mounted in uninterruptible mode using the **-o nointr** option.

on-db-partitions-clause
Specifies the database partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the database partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clauses*. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the *on-db-partitions-clause* is not specified, the containers will also be created on all new database partitions added to the database.

MANAGED BY DATABASE
Specifies that the table space is to be a DMS table space. When the type of table space is not specified, the default behavior is to create a large table space.

MANAGED BY DATABASE cannot be specified in a DB2 pureScale environment (SQLSTATE 42997).

Important: Starting with Version 10.1 Fix Pack 1, the DMS table space type is deprecated for user-defined permanent table spaces and might be removed in a

future release. The DMS table space type is not deprecated for catalog and temporary table spaces. For more information, see "DMS permanent table spaces have been deprecated" in *What's New for DB2 Version 10.1*.

database-containers
Specify the containers for a DMS table space.

USING
Introduces a container-clause.

container-clause
Specifies the containers for a DMS table space.

(FILE|DEVICE *'container-string' number-of-pages,...***)**
For a DMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The type of the container (either FILE or DEVICE) and its size (in PAGESIZE pages) are specified. The size can also be specified as an integer value followed by K (for kilobytes), M (for megabytes) or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the pagesize is used to determine the number of pages for the container. A mixture of FILE and DEVICE containers can be specified. The *container-string* cannot exceed 254 bytes in length.

For a FILE container, *container-string* must be an absolute or relative file name. The file name, if not absolute, is relative to the database directory. If any component of the directory name does not exist, it is created by the database manager. If the file does not exist, it will be created and initialized to the specified size by the database manager. When a table space is dropped, all components created by the database manager are deleted.

Note: If the file exists, it is overwritten, and if it is smaller than specified, it is extended. The file will not be truncated if it is larger than specified.

For a DEVICE container, *container-string* must be a device name. The device must already exist.

All containers must be unique across all databases. A container can belong to only one table space. The size of the containers can differ; however, optimal performance is achieved when all containers are the same size. The exact format of *container-string* is dependent on the operating system.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only supported with the use of an uninterrupted power supply (UPS); continuous UPS (rather than standby) is recommended.

on-db-partitions-clause
Specifies the database partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the database partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clause*. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the

on-db-partitions-clause is not specified, the containers will also be created on all new database partitions added to the database.

on-db-partitions-clause
Specifies the database partitions on which containers are created in a partitioned database.

ON DBPARTITIONNUMS
Keywords indicating that individual database partitions are specified. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1
Specify a database partition number.

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). Containers are to be created on each database partition between and including the specified values. A specified database partition must be in the database partition group for the table space.

The database partition specified by number, and every database partition within the specified range of database partitions must exist in the database partition group for the table space (SQLSTATE 42729). A database partition number can only appear explicitly or within a range in exactly one *on-db-partitions-clause* for the statement (SQLSTATE 42613).

EXTENTSIZE *number-of-pages*
Specifies the number of PAGESIZE pages that will be written to a container before skipping to the next container. The extent size value can also be specified as an integer value followed by K (for kilobytes) or M (for megabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the value for the extent size. The database manager cycles repeatedly through the containers as data is stored.

In a DB2 pureScale environment, you should use an extent size of at least 32 pages. This minimum extent size reduces the amount of internal message traffic within the DB2 pureScale environment when extents are added for a table or index.

The default value is provided by the **dft_extent_sz** database configuration parameter, which has a valid range of 2-256 pages.

PREFETCHSIZE
Specifies to read in data needed by a query before it being referenced by the query, so that the query need not wait for I/O to be performed.

The default value is provided by the **dft_prefetch_sz** database configuration parameter.

AUTOMATIC
Specifies that the prefetch size of a table space is to be updated automatically; that is, the prefetch size will be managed by the DB2 database manager.

DB2 will update the prefetch size automatically whenever the number of containers in a table space changes (following successful execution

of an ALTER TABLESPACE statement that adds or drops one or more containers). The prefetch size is also automatically updated at database startup.

number-of-pages
Specifies the number of PAGESIZE pages that will be read from the table space when data prefetching is being performed. The maximum value is 32767.

integer **K | M**
Specifies the prefetch size value as an integer value followed by K (for kilobytes) or M (for megabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the number of pages value for prefetch size.

BUFFERPOOL *bufferpool-name*
The name of the buffer pool used for tables in this table space. The buffer pool must exist (SQLSTATE 42704). If not specified, the default buffer pool (IBMDEFAULTBP) is used. The page size of the buffer pool must match the page size specified (or defaulted) for the table space (SQLSTATE 428CB). The database partition group of the table space must be defined for the buffer pool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds* **or OVERHEAD INHERIT**
Specifies the I/O controller overhead and disk seek and latency time. This value is used to determine the cost of I/O during query optimization. If OVERHEAD is not specified for a non-automatic storage table space, the value will default to the database creation default described later in the description for this keyword. If OVERHEAD is not specified for an automatic storage table space the default is to INHERIT the value from the storage group it is using. If the OVERHEAD value at the storage group is undefined, the OVERHEAD will default to the database creation default.

number-of-milliseconds
The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

INHERIT
If INHERIT is specified, the table space must be defined using automatic storage and the OVERHEAD is dynamically inherited from the storage group. INHERIT cannot be specified if the table space is not defined using automatic storage (SQLSTATE 42613).

For a database that was created in DB2 V10.1 or later, the default I/O controller overhead and disk seek and latency time for 4 KB PAGESIZE table space is 6.725 milliseconds.

For a database that was upgraded from a previous version of DB2 to DB2 V10.1 or later, the default I/O controller overhead and disk seek and latency time is as follows:

- 7.5 milliseconds for a database created in DB2 version 9.1 or higher
- 12.67 milliseconds for databases created between DB2 version 8.2 and DB2 version 9.1
- 24.1 milliseconds for DB2 versions previous to 8.2

FILE SYSTEM CACHING or NO FILE SYSTEM CACHING
Specifies whether or not I/O operations are to be cached at the file system level. If neither option is specified, the default is:

- FILE SYSTEM CACHING for JFS on AIX, Linux System z®, all non-VxFS file systems on Solaris, HP-UX, SMS temporary table space files on all platforms, and all LOB and large data
- NO FILE SYSTEM CACHING on all other platforms and file system types

FILE SYSTEM CACHING
Specifies that all I/O operations in the target table space are to be cached at the file system level.

NO FILE SYSTEM CACHING
Specifies that all I/O operations are to bypass the file system-level cache.

TRANSFERRATE *number-of-milliseconds* **or TRANSFERRATE INHERIT**
Specifies the time to read one page into memory. If TRANSFERRATE is not specified for a non-automatic storage table space, the value will default to the database creation default described later in the description for this keyword. If TRANSFERRATE is not specified for an automatic storage table space the default is to INHERIT the value from the storage group it is using. If the DEVICE READ RATE value at the storage group is undefined, the TRANSFERRATE will default to the database creation default.

number-of-milliseconds
This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

INHERIT
If INHERIT is specified, the table space must be defined using automatic storage and the TRANSFERRATE is dynamically inherited from the DEVICE READ RATE of the storage group. INHERIT cannot be specified if the table space is not defined using automatic storage (SQLSTATE 42613).

When an automatic storage table space inherits the TRANSFERRATE setting from the storage group it is using, the DEVICE READ RATE of the storage group, which is in megabytes per second, is converted into milliseconds per page read accounting for the PAGESIZE setting of the table space. The conversion formula follows:

TRANSFERRATE = (1 / DEVICE READ RATE) * 1000 / 1024000 * PAGESIZE

For a database that was created in DB2 V10.1 or later, the default time to read one page into memory for 4 KB PAGESIZE table space is 0.04 milliseconds.

For a database that was upgraded from a previous version of DB2 to DB2 V10.1 or later, the default time to read one page into memory is as follows:

- 0.06 milliseconds for a database created in DB2 version 9.1 or higher
- 0.18 milliseconds for databases created between DB2 version 8.2 or version 9.1
- 0.9 milliseconds for DB2 versions previous to 8.2

DATA TAG *integer-constant*, **DATA TAG INHERIT or DATA TAG NONE**
Specifies a tag for the data in the table space. If the DATA TAG is not specified, the default for automatic storage table spaces is to INHERIT from the storage group it is using and for non-automatic table spaces it will be set to NONE. This value can be used as part of a WLM configuration in a work class definition or referenced within a threshold definition; for more information refer to the CREATE WORK CLASS SET and CREATE THRESHOLD statements. This clause cannot be specified if TEMPORARY is also specified (SQLSTATE 42613).

integer-constant
Valid values for *integer-constant* are integers from 1 to 9. If an *integer-constant* is specified and there is an associated storage group, the data tag specified for the table space will override any data tag value specified for the associated storage group.

INHERIT
If INHERIT is specified, the table space must be defined using automatic storage and the data tag is dynamically inherited from the storage group. INHERIT cannot be specified if the table space is not defined using automatic storage (SQLSTATE 42613).

NONE
If NONE is specified, there is no data tag.

DROPPED TABLE RECOVERY
Indicates whether dropped tables in the specified table space can be recovered using the **RECOVER DROPPED TABLE** option of the **ROLLFORWARD DATABASE** command. This clause can only be specified for a regular or large table space (SQLSTATE 42613).

ON Specifies that dropped tables can be recovered. This has been the default since Version 8.

OFF
Specifies that dropped tables cannot be recovered. This is the default in Version 7.

Rules

- If automatic storage is not defined for the database, an error is returned (SQLSTATE 55060).
- The INITIALSIZE clause cannot be specified with the MANAGED BY SYSTEM or MANAGED BY DATABASE clause (SQLSTATE 42601).
- The AUTORESIZE, INCREASESIZE, or MAXSIZE clause cannot be specified with the MANAGED BY SYSTEM clause (SQLSTATE 42601).
- The AUTORESIZE, INITIALSIZE, INCREASESIZE, or MAXSIZE clause cannot be specified for the creation of a temporary automatic storage table space (SQLSTATE 42601).
- The INCREASESIZE or MAXSIZE clause cannot be specified if the tables space is not auto-resizable (SQLSTATE 42601).
- AUTORESIZE cannot be enabled for DMS table spaces that are defined to use raw device containers (SQLSTATE 42601).
- A table space must initially be large enough to hold five extents (SQLSTATE 57011).
- The maximum size of a table space must be larger than its initial size (SQLSTATE 560B0).

- Container operations (ADD, EXTEND, RESIZE, DROP, or BEGIN NEW STRIPE SET) cannot be performed on automatic storage table spaces, because the database manager is controlling the space management of such table spaces (SQLSTATE 42858).
- Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container definitions for the table space cannot exceed 208 kilobytes (SQLSTATE 54034).
- For a partitioned database, if more than one database partition resides on the same physical node, the same device or path cannot be specified for more than one database partition (SQLSTATE 42730). In this environment, either specify a unique *container-string* for each database partition, or use a relative path name.
- Only automatic storage table spaces can be created in a DB2 pureScale environment. (SQLSTATE 42997)
- ***Container size limits:*** In DMS table spaces, a container must be at least two times the extent size pages in length (SQLSTATE 54039). The maximum size of a container is operating system dependent.

Notes

- Choosing between a database-managed space or a system-managed space for a table space is a fundamental choice involving trade-offs.
- When more than one TEMPORARY table space exists in the database, they are used in round-robin fashion to balance their usage.
- The owner of the table space is granted USE privilege with the WITH GRANT OPTION on the table space when it is created.
- An automatic storage table space is created as either an SMS table space or a DMS table space. DMS is chosen for large and regular table spaces, and SMS is chosen for temporary table spaces. Note that this behavior cannot be depended upon, because it might change in a future release. When DMS is chosen and the type of table space is not specified, the default behavior is to create a large table space.
- The creation of an automatic storage table space does not include container definitions. The database manager automatically determines the location and size, if applicable, of the containers on the basis of the storage paths that are associated with the specified storage group or the default storage group. The database manager will attempt to grow large and regular table spaces, as necessary, provided that the maximum size has not been reached. This might involve extending existing containers or adding containers to a new stripe set. Every time that the database is activated, the database manager automatically reconfigures the number and location of the containers for temporary table spaces that are not in an abnormal state.
- A large or regular automatic storage table space will not use new storage paths (see the description of the ALTER STOGROUP statement) until there is no more space in one of the existing storage paths that the table space is using. Temporary automatic storage table spaces can only use the new storage paths once the database has been deactivated and then reactivated.
- ***Media attributes*:** The following table shows how the media attributes of newly created table spaces are treated in upgraded and newly created DB2 V10.1 databases.

Table 25. Media attributes across different versions of DB2

Media attributes	Upgraded Database	Newly Created Database
New automatic storage table spaces / storage group DEVICE READ RATE set to *undefined*	Defaults based on version database was created (no change)	Not applicable
New automatic storage table spaces / storage group OVERHEAD set to *undefined*	Defaults based on version database was created (no change)	Not applicable
New automatic storage table spaces / storage group DEVICE READ RATE is set	Inherit from storage group factoring in PAGESIZE	Inherit from storage group factoring in PAGESIZE
New automatic storage table spaces / storage group OVERHEAD is set	Inherit from storage group	Inherit from storage group
New non-automatic storage table spaces	Defaults based on version database was created (no change)	DB2 V10.1 media defaults taking PAGESIZE into account

- ***Default TRANSFERRATE in DB2 V10.1***: The following table shows how the default TRANSFERRATE value differs for newly created table spaces in DB2 V10.1.

Table 26. Default TRANSFERRATE in DB2 V10.1

PAGESIZE	TRANSFERRATE
4 KB	0.04 ms per page read
8 KB	0.08 ms per page read
16 KB	0.16 ms per page read
32 KB	0.32 ms per page read

- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NODE can be specified in place of DBPARTITIONNUM
  - NODES can be specified in place of DBPARTITIONNUMS
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
  - LONG can be specified in place of LARGE

Examples

- *Example 1:* Create a large DMS table space on a UNIX system using three devices of 10 000 4K pages each. Specify their I/O characteristics.

```
CREATE TABLESPACE PAYROLL
  MANAGED BY DATABASE
  USING (DEVICE'/dev/rhdisk6' 10000,
    DEVICE '/dev/rhdisk7' 10000,
    DEVICE '/dev/rhdisk8' 10000)
  OVERHEAD 12.67
  TRANSFERRATE 0.18
```

- *Example 2:* Create a regular SMS table space on Windows using three directories on three separate drives, with a 64-page extent size, and a 32-page prefetch size.

```
CREATE TABLESPACE ACCOUNTING
  MANAGED BY SYSTEM
  USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
  EXTENTSIZE 64
  PREFETCHSIZE 32
```

- *Example 3:* Create a system temporary DMS table space on a UNIX system using two files of 50 000 pages each, and a 256-page extent size.

```
CREATE TEMPORARY TABLESPACE TEMPSPACE2
  MANAGED BY DATABASE
  USING (FILE 'dbtmp/tempspace2.f1' 50000,
    FILE 'dbtmp/tempspace2.f2' 50000)
  EXTENTSIZE 256
```

- *Example 4:* Create a large DMS table space in database partition group ODDNODEGROUP (database partitions 1, 3, and 5) on a UNIX system. Use the device `/dev/rhdisk0` for 10 000 4K pages on each database partition. Specify a database partition-specific device with 40 000 4K pages for each database partition.

```
CREATE TABLESPACE PLANS
  MANAGED BY DATABASE
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn1hd01' 40000)
  ON DBPARTITIONNUM (1)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn3hd03' 40000)
  ON DBPARTITIONNUM (3)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn5hd05' 40000)
  ON DBPARTITIONNUM (5)
```

- *Example 5:* Create a large automatic storage table space named DATATS, allowing the system to make all decisions with respect to table space size and growth.

```
CREATE TABLESPACE DATATS
```

  or

```
CREATE TABLESPACE DATATS
  MANAGED BY AUTOMATIC STORAGE
```

- *Example 6:* Create a system temporary automatic storage table space named TEMPDATA.

```
CREATE TEMPORARY TABLESPACE TEMPDATA
```

  or

```
CREATE TEMPORARY TABLESPACE TEMPDATA
  MANAGED BY AUTOMATIC STORAGE
```

- *Example 7:* Create a large automatic storage table space named USERSPACE3 with an initial size of 100 megabytes and a maximum size of 1 gigabyte.

```
CREATE TABLESPACE USERSPACE3
  INITIALSIZE 100 M
  MAXSIZE 1 G
```

- *Example 8:* Create a large automatic storage table space named LARGEDATA with a growth rate of 10 percent (that is, its total size increases by 10 percent each time that it is automatically resized) and a maximum size of 512 megabytes. Instead of specifying the INITIALSIZE clause, let the database manager determine an appropriate initial size for the table space.

```
CREATE LARGE TABLESPACE LARGEDATA
  INCREASESIZE 10 PERCENT
  MAXSIZE 512 M
```

- *Example 9:* Create a large DMS table space named USERSPACE4 with two file containers (each container being 1 megabyte in size), a growth rate of 2 megabytes, and a maximum size of 100 megabytes.

```
CREATE TABLESPACE USERSPACE4
  MANAGED BY DATABASE USING (FILE '/db2/file1' 1 M, FILE '/db2/file2' 1 M)
  AUTORESIZE YES
  INCREASESIZE 2 M
  MAXSIZE 100 M
```

- *Example 10:* Create large DMS table spaces, using RAW devices on a Windows operating system.
  - To specify entire physical drives, use the \\.*physical-drive* format:

    ```
    CREATE TABLESPACE TS1
      MANAGED BY DATABASE USING (DEVICE '\\.\PhysicalDrive5' 10000,
        DEVICE '\\.\PhysicalDrive6' 10000)
    ```

  - To specify logical partitions by using drive letters:

    ```
    CREATE TABLESPACE TS2
      MANAGED BY DATABASE USING (DEVICE '\\.\G:' 10000,
        DEVICE '\\.\H:' 10000)
    ```

  - To specify logical partitions by using volume global unique identifiers (GUIDs), use the **db2listvolumes** utility to retrieve the volume GUID for each local partition, then copy the GUID for the logical partition that you want into the table space container clause:

    ```
    CREATE TABLESPACE TS3
      MANAGED BY DATABASE USING (
        DEVICE '\\?\Volume{2ca6a0c1-8542-11d8-9734-00096b5322d2}\' 20000M)
    ```

    You might prefer to use volume GUIDs over the drive letter format if you have more partitions than available drive letters on the machine.

  - To specify logical partitions by using junction points (or volume mount points), mount the RAW partition to another NTFS-formatted volume as a junction point, then specify the path to the junction point on the NTFS volume as the container path. For example:

    ```
    CREATE TABLESPACE TS4
      MANAGED BY DATABASE USING (DEVICE 'C:\JUNCTION\DISK_1' 10000,
        DEVICE 'C:\JUNCTION\DISK_2' 10000)
    ```

    DB2 first queries the partition to see whether there is a file system on it; if yes, the partition is not treated as a RAW device, and DB2 performs normal file system I/O operations on the partition.

CREATE THRESHOLD

The CREATE THRESHOLD statement defines a threshold.

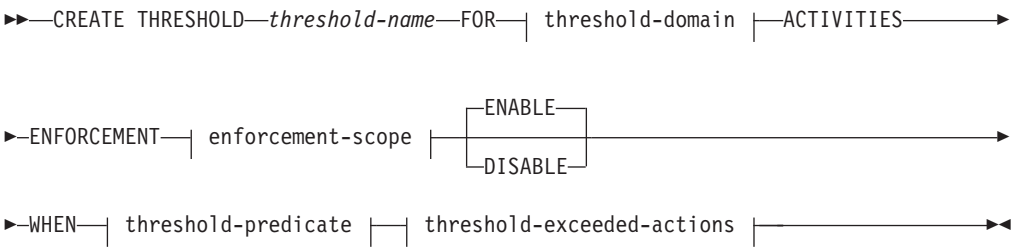
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

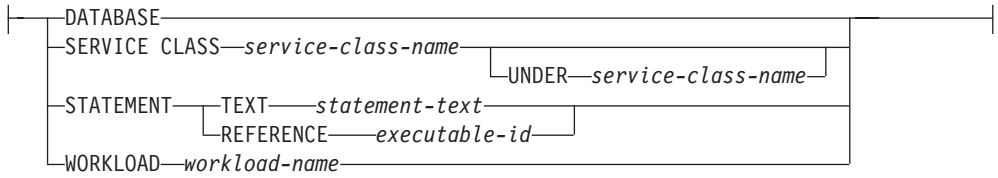
Authorization

The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

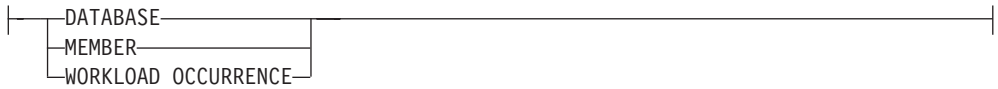
Syntax



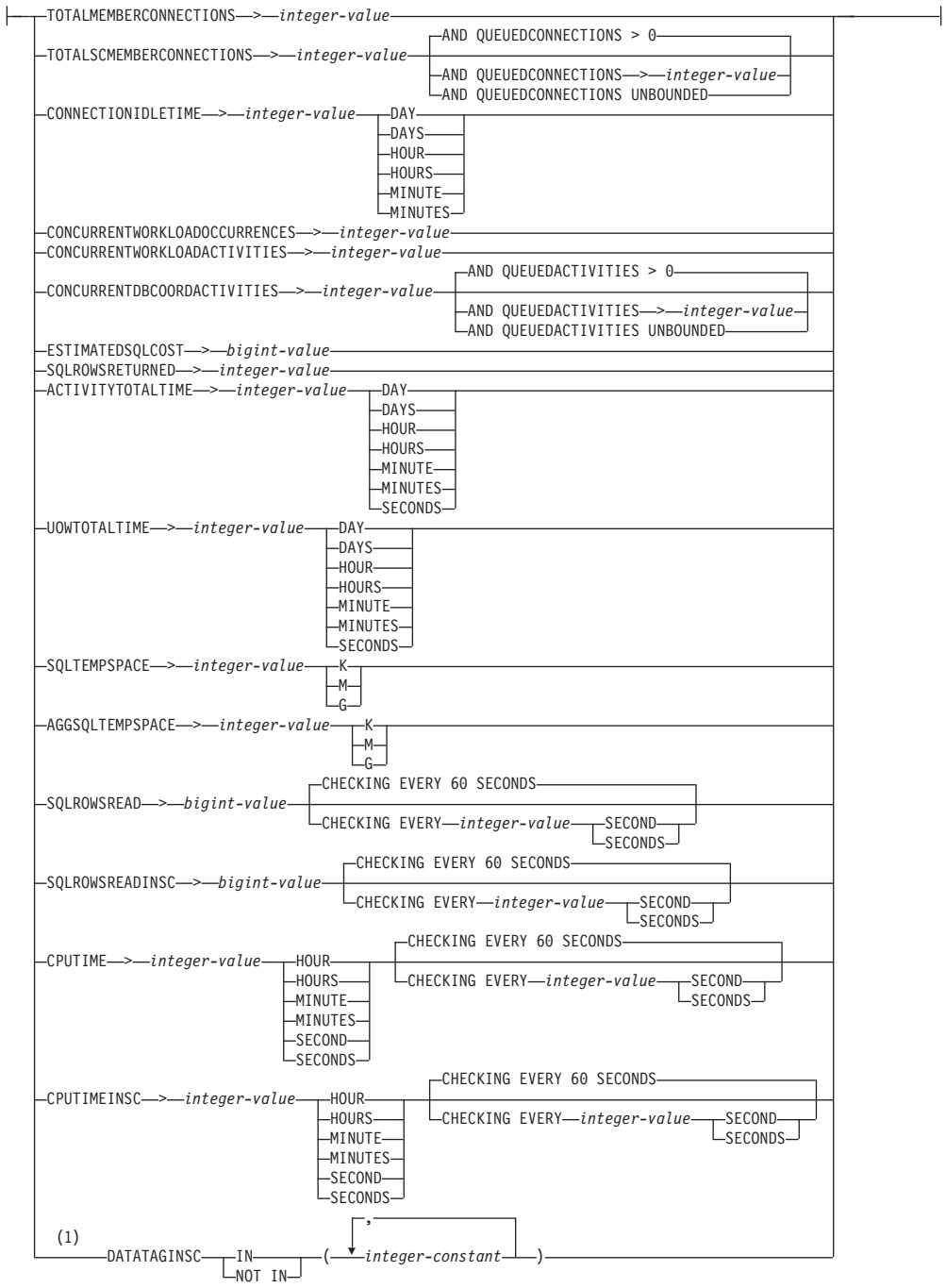
threshold-domain:



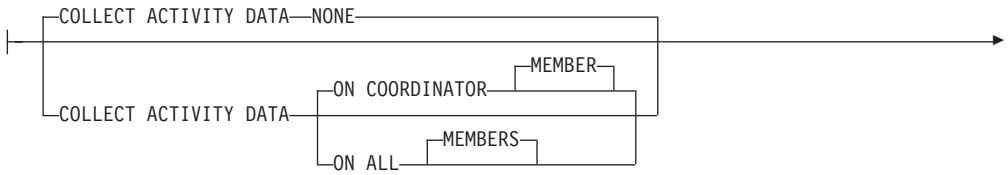
enforcement-scope:

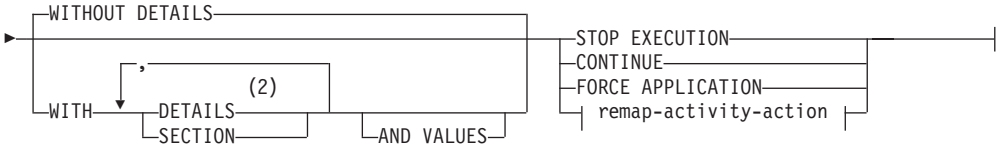


threshold-predicate:

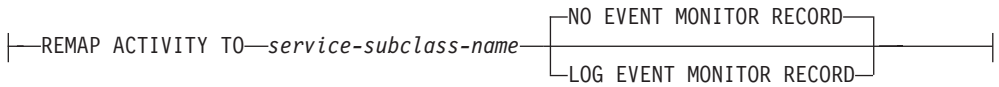


threshold-exceeded-actions:





remap-activity-action:



Notes:

1 Each data tag value can be specified only once.

2 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

Description

threshold-name
Names the threshold. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *threshold-name* must not identify a threshold that already exists at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

FOR *threshold-domain* **ACTIVITIES**
Specifies the definition domain of the threshold.

DATABASE
This threshold applies to any activity in the database.

SERVICE CLASS *service-class-name*
This threshold applies to activities executing in service class *service-class-name*. If UNDER is not specified, *service-class-name* must identify an existing service superclass (SQLSTATE 42704). If UNDER is specified, *service-class-name* must identify an existing service subclass of the service superclass specified after the UNDER keyword (SQLSTATE 42704). The *service-class-name* cannot be the SYSDEFAULTSYSTEMCLASS service class or the SYSDEFAULTMAINTENANCECLASS service class (SQLSTATE 5U032).

UNDER *service-class-name*
Specifies a service superclass. The *service-class-name* must identify an existing service superclass (SQLSTATE 42704).

STATEMENT
This threshold applies to activities for a specific SQL statement. You identify the statement to use for the threshold by specifying the statement text or the statement's executable ID .

TEXT *statement-text*
This threshold applies to statements matching the text specified in *statement-text*. Both static and dynamic SQL statements are considered when the condition for the threshold is evaluated. At run time, the text specified for *statement-text* must be an exact match of the text of a statement in the package cache for the threshold to be violated. Differences in letter case or use of white space prevent a match from occurring between *statement-text* and any running SQL statement. The

text for *statement-text* must be specified as a string constant. As such, the maximum length for the text of a statement for a statement threshold is 32 672 bytes, and not the usual 2 MB upper limit for statements.

Access plan differences do not affect statement matching. It is possible for multiple cached statements with same text but different access plans to match the threshold text defined by *statement-text*.

If a statement that otherwise matches the statement supplied for *statement-text* is altered or transformed during compilation in such a way that it differs from *statement-text*, the statements will not match. For example, if the statement concentrator is enabled, literal values might be replaced by parameter markers. No such transformation is applied to text supplied for the *statement-text* in the CREATE THRESHOLD statement. The text supplied to CREATE THRESHOLD must match exactly the transformed text of any statement of interest. You can determine the exact text of statements as they are executed using monitoring table functions such as **MON_GET_PKG_CACHE_STMT** and **MON_GET_ACTIVITY_DETAILS**.

The following predicates can be used with a statement threshold: **ESTIMATEDSQLCOST**, **SQLROWSRETURNED**, **ACTIVITYTOTALTIME**, **SQLROWSREAD**, **CPUTIME**, **SQLTEMPSPACE**.

REFERENCE *executable-id*
This threshold applies to statements with text that matches the text of the statement with the specified executable ID. The database manager uses the executable ID to locate text of the statement from its section in the package cache. The text of the statement that is used for the threshold is that which was cached for the section at the time the threshold was created. For dynamic SQL, the statement referenced by the executable ID must be in the package cache. For static SQL, if the statement is not in the cache, the database manager retrieves it from the system catalogs.

Once the statement text is retrieved from the package cache, there is no direct relationship between the threshold and the specified executable ID; the cached section can even be evicted from the cache without impact on any threshold that was derived from it. Once the text associated with the executable ID is determined, the threshold created by this clause behaves in exactly the same way as one created by the STATEMENT TEXT clause.

WORKLOAD *workload-name*
This threshold applies to the specified workload. The *workload-name* must identify an existing workload (SQLSTATE 42704).

ENFORCEMENT *enforcement-scope*
The enforcement scope of the threshold.

DATABASE
The threshold is enforced across all members within the definition domain; that is, all members of the database, and all members of the service class.

MEMBER
The threshold is enforced on a per member basis. There is no coordination across all members to enforce the threshold.

WORKLOAD OCCURRENCE
The threshold is enforced only within a workload occurrence. Two

workload occurrences running concurrently on the same member will each have their own running count for this threshold.

ENABLE or DISABLE
Specifies whether or not the threshold is enabled for use by the database manager.

ENABLE
The threshold is used by the database manager to restrict the execution of database activities.

DISABLE
The threshold is not used by the database manager to restrict the execution of database activities.

WHEN *threshold-predicate*
Specifies the condition of the threshold.

TOTALMEMBERCONNECTIONS > *integer-value*
This condition defines an upper bound on the number of coordinator connections that can run concurrently on a member. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that any new coordinator connection will be prevented from connecting. All currently running or queued connections will continue. The definition domain for this condition must be DATABASE, and the enforcement scope must be MEMBER (SQLSTATE 5U037). This threshold is not enforced for users with DBADM or WLMADM authority.

TOTALSCMEMBERCONNECTIONS > *integer-value*
This condition defines an upper bound on the number of coordinator connections that can run concurrently on a member in a specific service superclass. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that any new connection will be prevented from joining the service class. All currently running or queued connections will continue. The definition domain for this condition must be SERVICE SUPERCLASS, and the enforcement scope must be MEMBER (SQLSTATE 5U037).

AND QUEUEDCONNECTIONS > *integer-value* **or AND QUEUEDCONNECTIONS UNBOUNDED**
Specifies a queue size for when the maximum number of coordinator connections is exceeded. This value can be any positive integer, including zero (SQLSTATE 42820). A value of zero means that no coordinator connections are queued. Specifying UNBOUNDED will queue every connection that exceeds the specified maximum number of coordinator connections, and the *threshold-exceeded-actions* will never be executed. The default is zero.

CONNECTIONIDLETIME > *integer-value* **DAY | DAYS | HOUR | HOURS | MINUTE | MINUTES**
This condition defines an upper bound for the amount of time the database manager will allow a connection to remain idle. This value can be any positive integer (not zero) (SQLSTATE 42820). Use a valid duration keyword to specify an appropriate unit of time for *integer-value*. The definition domain for this condition must be DATABASE or SERVICE SUPERCLASS, and the enforcement scope must be DATABASE (SQLSTATE 5U037). This condition is enforced at the coordinator member.

If you specify the STOP EXECUTION action with CONNECTIONIDLETIME thresholds, the connection for the application is

dropped when the threshold is exceeded. Any subsequent attempt by the application to access the data server will not receive SQLSTATE 5U026.

The maximum value for this threshold is 2 147 483 640 seconds. Any value specified that has a seconds equivalent larger than 2 147 483 640 seconds will be set to this number of seconds.

CONCURRENTWORKLOADOCCURRENCES > *integer-value*
This condition defines an upper bound on the number of concurrent occurrences for the workload on each member. This value can be any positive integer (not zero) (SQLSTATE 42820). The definition domain for this condition must be WORKLOAD and the enforcement scope must be MEMBER (SQLSTATE 5U037).

CONCURRENTWORKLOADACTIVITIES > *integer-value*
This condition defines an upper bound on the number of concurrent coordinator activities and nested activities for the workload on each member. This value can be any positive integer (not zero) (SQLSTATE 42820). The definition domain for this condition must be WORKLOAD and the enforcement scope for this condition must be WORKLOAD OCCURRENCE (SQLSTATE 5U037).

Each nested activity must satisfy the following conditions:

- It must be a recognized coordinator activity. Any nested coordinator activity that does not fall within the recognized types of activities will not be counted. Similarly, nested subagent activities, such as remote node requests, are not counted.
- It must be directly invoked from user logic, such as a user-written procedure issuing SQL statements.

Consequently, nested coordinator activities that were automatically started under the invocation of a DB2 utility or routines in the SYSIBM, SYSFUN, or SYSPROC schemas are not counted toward the upper bound specified by this threshold.

Internal SQL activities, such as those initiated by the setting of a constraint or the refreshing of a materialized query table, are also not counted by this threshold, because they are initiated by the database manager and not directly invoked by user logic.

CONCURRENTDBCOORDACTIVITIES > *integer-value*
This condition defines an upper bound on the number of recognized database coordinator activities that can run concurrently on all members in the specified domain. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that any new database coordinator activities will be prevented from executing. All currently running or queued database coordinator activities will continue. The definition domain for this condition must be DATABASE, work action (a threshold for a work action definition domain is created using a CREATE WORK ACTION SET or ALTER WORK ACTION SET statement, and the work action set must be applied to a workload or a database), SERVICE SUPERCLASS, or SERVICE SUBCLASS. Also, the enforcement scope must be DATABASE (SQLSTATE 5U037) in environments other than DB2 pureScale, where the condition is enforced across the entire database, and MEMBER (SQLSTATE 5U037) in DB2 pureScale where the condition is enforced at each coordinator member. All activities are tracked by this condition, except for the following items:

- CALL statements are not controlled by this threshold, but all nested child activities started within the called routine are under this threshold's control. Anonymous blocks and autonomous routines are classified as CALL statements.
- User-defined functions are controlled by this threshold, but child activities nested in a user-defined function are not controlled. If an autonomous routine is called from within a user defined function, neither the autonomous routine nor any child activities of the autonomous routine are under threshold control.
- Trigger actions that invoke CALL statements and the child activities of these CALL statements are not controlled by this threshold. INSERT, UPDATE, or DELETE statements that can cause a trigger to activate continue to be under threshold control.

When a threshold is defined as part of a work action set, the enforcement scope is determined automatically based on the current environment (MEMBER, if the current environment is DB2 pureScale; DATABASE, if it is otherwise).

Important: Before using CONCURRENTDBCOORDACTIVITIES thresholds, be sure to become familiar with the effects that they can have on the database system. For more information, see the "CONCURRENTDBCOORDACTIVITIES threshold" topic.

AND QUEUEDACTIVITIES > *integer-value* **or AND QUEUEDACTIVITIES UNBOUNDED**

Specifies a queue size for when the maximum number of database coordinator activities is exceeded. This value can be zero or any positive integer (SQLSTATE 42820). A value of zero means that no database coordinator activities are queued. Specifying UNBOUNDED will queue every database coordinator activity that exceeds the specified maximum number of database coordinator activities, and the *threshold-exceeded-actions* will never be executed. The default is zero.

Note: If a threshold action of CONTINUE is specified for a queuing threshold, it effectively makes the size of the queue unbounded, regardless of any hard value specified for the queue size.

ESTIMATEDSQLCOST > *bigint-value*

This condition defines an upper bound for the optimizer-assigned cost (in timerons) of an activity. This value can be any positive big integer (not zero) (SQLSTATE 42820). The definition domain for this condition must be DATABASE, work action (a threshold for a work action definition domain is created using a CREATE WORK ACTION SET or ALTER WORK ACTION SET statement, and the work action set must be applied to a workload or a database), SERVICE SUPERCLASS, SERVICE SUBCLASS, or WORKLOAD, and the enforcement scope must be DATABASE (SQLSTATE 5U037). This condition is enforced at the coordinator member. Activities tracked by this condition are:

- Coordinator activities of type data manipulation language (DML).
- Nested DML activities that are invoked from user logic. Consequently, DML activities that can be initiated by the database manager (such as utilities, procedures, or internal SQL) are not tracked by this condition (unless their cost is included in the parent's estimate, in which case they are indirectly tracked).

SQLROWSRETURNED > *integer-value*
This condition defines an upper bound for the number of rows returned to a client application from the application server. This value can be any positive integer (not zero) (SQLSTATE 42820). The definition domain for this condition must be DATABASE, work action (a threshold for a work action definition domain is created using a CREATE WORK ACTION SET or ALTER WORK ACTION SET statement, and the work action set must be applied to a workload or a database), SERVICE SUPERCLASS, SERVICE SUBCLASS, or WORKLOAD, and the enforcement scope must be DATABASE (SQLSTATE 5U037). This condition is enforced at the coordinator member. Activities tracked by this condition are:

- Coordinator activities of type DML.
- Nested DML activities that are derived from user logic. Activities that are initiated by the database manager through a utility, procedure, or internal SQL are not affected by this condition.

Result sets returned from within a procedure are treated separately as individual activities. There is no aggregation of the rows that are returned by the procedure itself.

ACTIVITYTOTALTIME > *integer-value* **DAY** | **DAYS** | **HOUR** | **HOURS** | **MINUTE** | **MINUTES** | **SECONDS**
This condition defines an upper bound for the amount of time the database manager will allow an activity to execute, including the time the activity was queued. The activities that are covered by this threshold include the execution of SQL statements, not including compilation time, and the load utility. This value can be any positive integer (not zero) (SQLSTATE 42820). Use a valid duration keyword to specify an appropriate unit of time for *integer-value*. The definition domain for this condition must be DATABASE, work action (a threshold for a work action definition domain is created using a CREATE WORK ACTION SET or ALTER WORK ACTION SET statement, and the work action set must be applied to a workload or a database), SERVICE SUPERCLASS, SERVICE SUBCLASS, or WORKLOAD, and the enforcement scope must be DATABASE (SQLSTATE 5U037). This condition is enforced at the coordinator member.

If the specified time unit is SECONDS, the value must be a multiple of 10 (SQLSTATE 42615). The maximum value that can be specified for this threshold is 2 147 483 640 seconds. Any value specified (using the DAY, HOUR, MINUTE, or SECONDS time unit) that has a seconds equivalent larger than 2 147 483 640 seconds will be truncated to this number of seconds.

UOWTOTALTIME > *integer-value* **DAY** | **DAYS** | **HOUR** | **HOURS** | **MINUTE** | **MINUTES** | **SECONDS**
This condition defines an upper bound for the amount of time the database manager will allow a unit of work to execute. This value can be any non-zero positive integer (SQLSTATE 42820). Use a valid duration keyword to specify an appropriate unit of time for *integer-value*. If the specified time unit is SECONDS, the value must be a multiple of 10 (SQLSTATE 42615). The definition domain for this condition must be DATABASE, SERVICE SUPERCLASS, or WORKLOAD, and the enforcement scope must be DATABASE (SQLSTATE 5U037). This condition is enforced at the coordinator member.

The maximum value that can be specified for this threshold is 2 147 483 640 seconds. If any value (using the DAY, HOUR, MINUTE, or SECONDS time unit) has a seconds equivalent larger than the maximum value, an error is returned (SQLSTATE 42615).

SQLTEMPSPACE > *integer-value* **K | M | G**

This condition defines the maximum amount of system temporary space that can be consumed by an SQL statement on a member. This value can be any positive integer (not zero) (SQLSTATE 42820).

If *integer-value K* (in either upper- or lowercase) is specified, the maximum size is 1024 times *integer-value*. If *integer-value M* is specified, the maximum size is 1 048 576 times *integer-value*. If *integer-value G* is specified, the maximum size is 1 073 741 824 times *integer-value*.

The definition domain for this condition must be DATABASE, work action (a threshold for a work action definition domain is created using a CREATE WORK ACTION SET or ALTER WORK ACTION SET statement, and the work action set must be applied to a workload or a database), SERVICE SUPERCLASS, SERVICE SUBCLASS, or WORKLOAD, and the enforcement scope must be MEMBER (SQLSTATE 5U037). Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (subsection execution). Activities that are initiated by the database manager through a utility, procedure, or internal SQL are not affected by this condition.

AGGSQLTEMPSPACE > *integer-value* **K | M | G**

This condition defines the maximum amount of system temporary space that can be consumed by a set of statements in a service class on a member. This value can be any positive integer (not zero) (SQLSTATE 42820).

If *integer-value* K (in either upper- or lowercase) is specified, the maximum size is 1024 times *integer-value*. If *integer-value* M is specified, the maximum size is 1 048 576 times *integer-value*. If *integer-value* G is specified, the maximum size is 1 073 741 824 times *integer-value*.

The definition domain for this condition must be SERVICE SUBCLASS and the enforcement scope must be MEMBER (SQLSTATE 5U037).

Activities contributing to the aggregate that is tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work like subsection execution.
- Nested DML activities that are derived from user logic and their corresponding subagent work like subsection execution. Activities initiated by the database manager through a utility, procedure, or internal SQL statement are not affected by this condition.

SQLROWSREAD > *bigint-value*

This condition defines an upper bound on the number of rows that may be read by an activity during its lifetime on a particular member. This value can be any positive big integer (not zero) (SQLSTATE 42820). Note that the number of rows read is different from the number of rows returned, which is controlled by the SQLROWSRETURNED condition.

The definition domain for this condition must be DATABASE, SERVICE CLASS, a service subclass (SERVICE CLASS specifying the UNDER clause), WORKLOAD or a work action (a threshold for a work action definition domain is created using a CREATE WORK ACTION SET or ALTER WORK ACTION SET statement, and the work action set must be applied to a workload or a database), and the enforcement scope must be MEMBER (SQLSTATE 5U037). This condition is enforced independently at each member.

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.
- Internal SQL activities like those initiated by the setting of a constraint, or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.

CHECKING EVERY *integer-value* **SECOND | SECONDS**
Specifies how frequently the threshold condition is checked for an activity. The threshold is checked at the end of each request (like a fetch operation, for example) and on the interval defined by the CHECKING clause. The CHECKING clause defines an upper bound on how long a threshold violation may go undetected. The default is 60 seconds. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820). Setting a low value may impact system performance negatively.

SQLROWSREADINSC > *bigint-value*
This condition defines an upper bound on the number of rows that may be read by an activity on a particular member while it is executing in a service subclass. Rows read before executing in the service subclass specified are not counted. This value can be any positive big integer (not zero) (SQLSTATE 42820). Note that the number of rows read is different from the number of rows returned, which is controlled by the SQLROWSRETURNED condition.

The definition domain for this condition must be a service subclass (SERVICE CLASS specifying the UNDER clause) and the enforcement scope must be MEMBER (SQLSTATE 5U037). This condition is enforced independently at each member.

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.

- Internal SQL activities like those initiated by the setting of a constraint, or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.

CHECKING EVERY *integer-value* **SECOND | SECONDS**
Specifies how frequently the threshold condition is checked for an activity. The threshold is checked at the end of each request (like a fetch operation, for example) and on the interval defined by the CHECKING clause. The CHECKING clause defines an upper bound on how long a threshold violation may go undetected. The default is 60 seconds. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820).Setting a low value may impact system performance negatively.

CPUTIME > *integer-value* **HOUR | HOURS | MINUTE | MINUTES | SECOND | SECONDS**
This condition defines an upper bound for the amount of processor time that an activity may consume during its lifetime on a particular member. The processor time tracked by this threshold is measured from the time that the activity starts executing. This value can be any positive integer (not zero) (SQLSTATE 42820).

The definition domain for this condition must be DATABASE, a service superclass (SERVICE CLASS), a service subclass (SERVICE CLASS specifying the UNDER clause), WORKLOAD or work action (a threshold for a work action definition domain is created using a CREATE WORK ACTION SET or ALTER WORK ACTION SET statement, and the work action set must be applied to a workload or a database), and the enforcement scope must be MEMBER (SQLSTATE 5U037). This condition is enforced independently at each member.

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.
- Internal SQL activities, like those initiated by the setting of a constraint or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.
- Activities of type CALL. For CALL activities, the processor time tracked for the procedure does not include the processor time used by any child activities or by any fenced mode processes. The threshold condition will be checked only upon return from user logic to the database engine. For example: During the execution of a trusted routine, the threshold condition will be checked only when the routine issues a request to the database engine).

CHECKING EVERY *integer-value* **SECOND | SECONDS**
Specifies how frequently the threshold condition is checked for an activity. The granularity of the CPUTIME threshold is approximately this number multiplied by the degree of parallelism

for the activity. For example: If the threshold is checked every 60 seconds and the degree of parallelism is 2, the activity might use an extra 2 minutes of processor time instead of 1 minute before the threshold violation is detected. The default is 60 seconds. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820). Setting a low value may impact system performance negatively.

CPUTIMEINSC > *integer-value* **HOUR | HOURS | MINUTE | MINUTES | SECOND | SECONDS**

This condition defines an upper bound for the amount of processor time that an activity may consume on a particular member while it is executing in a particular service subclass. The processor time tracked by this threshold is measured from the time that the activity starts executing in the service subclass identified in the threshold domain. Any processor time used before that point is not counted toward the limit imposed by this threshold. This value can be any positive integer (not zero) (SQLSTATE 42820).

The definition domain for this condition must be a service subclass (SERVICE CLASS specifying the UNDER clause), and the enforcement scope must be MEMBER (SQLSTATE 5U037). This condition is enforced independently at each member.

Activities tracked by this condition are:

- Coordinator activities of type DML and corresponding subagent work (like subsection execution).
- Nested DML activities that are derived from user logic and their corresponding subagent work (like subsection execution). Activities that are initiated by the database manager through a utility or procedure (with the exception of the ADMIN_CMD procedure) are not counted for this condition.
- Internal SQL activities, like those initiated by the setting of a constraint or the refreshing of a materialized query table, are also not tracked by this threshold, because they are initiated by the database manager and not directly invoked by user logic.
- Activities of type CALL. For CALL activities, the processor time tracked for the procedure does not include the processor time used by any child activities or by any fenced mode processes. The threshold condition will be checked only upon return from user logic to the database engine. For example: During the execution of a trusted routine, the threshold condition will be checked only when the routine issues a request to the database engine).

CHECKING EVERY *integer-value* **SECOND | SECONDS**

Specifies how frequently the threshold condition is checked for an activity. The granularity of the CPUTIMEINSC threshold is approximately this number multiplied by the degree of parallelism for the activity. For example: If the threshold is checked every 60 seconds and the degree of parallelism is 2, the activity might use an extra 2 minutes of processor time instead of 1 minute before the threshold violation is detected. The default is 60 seconds. The value can be any positive integer (not zero) with a maximum value of 86400 seconds (SQLSTATE 42820). Setting a low value may impact system performance negatively.

DATATAGINSC IN *(integer-constant, ...)*
This condition defines one or more data tag values specified on a table space that the activity touches. The data tag on a table space, or its underlying storage group (where applicable), can be either not be set or set to a value from 1 to 9. If the activity touches a table space that has no data tag set (at either the table space or the storage group level), this threshold will not have any affect on that activity. The definition domain for this condition must be a service subclass (SERVICE CLASS specifying the UNDER clause), and the enforcement scope must be DATABASE PARTITION (SQLSTATE 5U037). This condition is enforced independently at each database partition.

Activities tracked by this condition are:

- Coordinator activities of type data manipulation language (DML).
- Nested DML activities that are directly invoked from user logic.

DML activities that can be initiated by the database manager (such as utilities, procedures, or internal SQL) are not tracked by this condition.

This threshold is checked only when a scan is opened on a table or when an insert is performed into a table. Fetching data from a table after a scan has been opened will not violate the threshold.

DATATAGINSC NOT IN *(integer-constant, ...)*
This condition defines one or more data tag values not specified on a table space that the activity touches. The data tag on a table space, or its underlying storage group (where applicable), can be either not be set or set to a value from 1 to 9. If the activity touches a table space that has no data tag set (either at the table space or the storage group level), this threshold will not have any affect on that activity. The definition domain for this condition must be a service subclass (SERVICE CLASS specifying the UNDER clause) and the enforcement scope must be DATABASE PARTITION (SQLSTATE 5U037). This condition is enforced independently at each database partition.

Activities tracked by this condition are:

- Coordinator activities of type data manipulation language (DML).
- Nested DML activities that are directly invoked from user logic.

DML activities that can be initiated by the database manager (such as utilities, procedures, or internal SQL) are not tracked by this condition.

This threshold is checked only when a scan is opened on a table or when an insert is performed into a table. Fetching data from a table after a scan has been opened will not violate the threshold.

threshold-exceeded-actions
Specifies what action is to be taken when a condition is exceeded. Each time that a condition is exceeded, an event is recorded in the threshold violations event monitor, if one is active.

COLLECT ACTIVITY DATA
Specifies that data about each activity that exceeded the threshold is to be sent to any active activities event monitor, when the activity completes. The default is COLLECT ACTIVITY DATA NONE. If COLLECT ACTIVITY DATA is specified, the default is WITHOUT DETAILS. The COLLECT ACTIVITY DATA setting does not apply to non-activity thresholds, such as the following: CONNECTIONIDLETIME, TOTALDBPARTITIONCONNECTIONS,

TOTALSCPARTITIONCONNECTIONS,
CONCURRENTWORKLOADOCCURRENCES, UOWTOTALTIME.

NONE
Specifies that activity data should not be collected for each activity that exceeds the threshold.

ON COORDINATOR MEMBER
Specifies that the activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that the activity data is to be collected at all members on which the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. For predictive thresholds, activity information is collected at all members only if you also specify the CONTINUE action for exceeded thresholds. For reactive thresholds, the ON ALL MEMBERS clause has no effect and activity information is always collected only at the coordinator member. For both predictive and reactive thresholds, any activity details, section information, or values will be collected only at the coordinator member.

WITHOUT DETAILS
Specifies that data about each activity associated with the work class for which this work action is defined is to be sent to any active activities event monitor, when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. For predictive thresholds, section actuals will be collected on any member where the activity data is collected. For reactive thresholds, section actuals will be collected only on the coordinator member.

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

STOP EXECUTION
The execution of the activity is stopped and an error is returned (SQLSTATE 5U026). In the case of the UOWTOTALTIME threshold, the unit of work is rolled back.

CONTINUE
The execution of the activity is not stopped.

FORCE APPLICATION
The application is forced off the system (SQLSTATE 55032). This action can only be specified for the UOWTOTALTIME threshold.

remap-activity-action

REMAP ACTIVITY TO *service-subclass-name*
The activity is mapped to *service-subclass-name*.The execution of the activity is not stopped. This action is valid only for in-service-class thresholds like CPUTIMEINSC, SQLROWSREADINSC, DATATAGINSC IN and DATATAGINSC NOT IN thresholds (SQLSTATE 5U037). The service-subclass-name must identify an existing service subclass under the same superclass associated with the threshold (SQLSTATE 5U037). The service-subclass-name cannot be the same as the associated service subclass of the threshold (SQLSTATE 5U037).

NO EVENT MONITOR RECORD
Specifies that no threshold violation record will be written.

LOG EVENT MONITOR RECORD
Specifies that if a THRESHOLD VIOLATIONS event monitor exists and is active, a threshold violation record is written to it.

Notes

- Thresholds can be defined on different aspects of database behavior to monitor and control that behavior. When a threshold is defined on activities, unless otherwise specified, it will be enforced only during the actual execution of SQL statements, not including compilation time, and the load utility.
- The CONCURRENTWORKLOADOCCURRENCES threshold and the CONCURRENTWORKLOADACTIVITIES threshold differ in scope. CONCURRENTWORKLOADOCCURRENCES controls how many connections can map to a workload definition simultaneously, and CONCURRENTWORKLOADACTIVITIES controls how many activities each connection that is mapped to the workload definition can submit concurrently.
- Changes are written to the system catalog, but do not take effect until after a COMMIT statement, even for the connection that issues the statement.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.
- ***Threshold exceeded action of CONTINUE and event monitor data:*** Event monitor data is collected only once per member when a threshold condition has been exceeded. If the threshold exceeded action is CONTINUE, the activity continues executing and no further event monitor data is collected for that threshold at the affected member. For example, consider a time threshold of 10 minutes with an action of CONTINUE. After an activity exceeds the 10-minute upper bound, event monitor data is collected for the threshold at the affected member.
- ***Quiescing a service class:*** The TOTALSCPARTITIONCONNECTIONS threshold condition can be used to simulate quiescing service classes that cannot normally be quiesced (for example, the default user class, or the default system class). This is useful, because thresholds do not apply to users with DBADM authority running in the SYSDEFAULTADMWORKLOAD, whereas a quiesced service class is not available to anyone. Consequently, default service classes cannot be quiesced directly but only through a threshold that allows users with DBADM authority to join them when connected to the database using the SYSDEFAULTADMWORKLOAD.

- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - TOTALDBPARTITIONCONNECTIONS can be specified in place of TOTALMEMBERCONNECTIONS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - TOTALSCPARTITIONCONNECTIONS can be specified in place of TOTALSCMEMBERCONNECTIONS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

Examples

- *Example 1:* Create a threshold that enforces a maximum temporary table space usage of 50M (per database partition) to any activity in the database. Any activity that violates this threshold is to be stopped.

```
CREATE THRESHOLD DBMAX50MEGTEMPSPACE
  FOR DATABASE ACTIVITIES
  ENFORCEMENT MEMBER
  WHEN SQLTEMPSPACE > 50 M
  STOP EXECUTION
```

- *Example 2:* Create a second threshold to limit the default runtime of any activity in the database to a maximum of 1 hour. Any activity that violates this threshold is to be stopped.

```
CREATE THRESHOLD DBMAX1HOURRUNTIME
  FOR DATABASE ACTIVITIES
  ENFORCEMENT DATABASE
  WHEN ACTIVITYTOTALTIME > 1 HOUR
  STOP EXECUTION
```

- *Example 3:* Assume that a service superclass named BIGQUERIES was created to host queries using more temporary space than average and running longer than 1 hour. The thresholds defined inside this service class will override the values that were set in the previous example at the database level. Note how activities violating the thresholds inside this superclass are allowed to continue executing, but detailed information is collected for further analysis.

```
CREATE THRESHOLD BIGQUERIESMAX500MEGTEMPSPACE
  FOR SERVICE CLASS BIGQUERIES ACTIVITIES
  ENFORCEMENT DATABASE MEMBER
  WHEN SQLTEMPSPACE  > 500 M
  COLLECT ACTIVITY DATA WITH DETAILS AND VALUES
  CONTINUE

CREATE THRESHOLD BIGQUERIESLONGRUNNINGTIME
  FOR SERVICE CLASS BIGQUERIES ACTIVITIES
  ENFORCEMENT DATABASE
  WHEN ACTIVITYTOTALTIME   > 10 HOURS
  COLLECT ACTIVITY DATA WITH DETAILS AND VALUES
  CONTINUE
```

- *Example 4:* Assuming the existence of a workload named PAYROLL, create a threshold that enforces the maximum number of activities within the workload to be less than or equal to 10.

```
CREATE THRESHOLD MAXACTIVITIESINPAYROLL
  FOR WORKLOAD PAYROLL ACTIVITIES
  ENFORCEMENT WORKLOAD OCCURRENCE
  WHEN CONCURRENTWORKLOADACTIVITIES > 10
  STOP EXECUTION
```

- *Example 5:* Create a threshold that enforces a maximum concurrency of 2 activities in the service class BIGQUERIES.

```
CREATE THRESHOLD MAXBIGQUERIESCONCURRENCY
  FOR SERVICE CLASS BIGQUERIES  ACTIVITIES
  ENFORCEMENT DATABASE
  WHEN CONCURRENTDBCOORDACTIVITIES > 2
  STOP EXECUTION
```

- *Example 6:* Create a threshold that captures activity information for a specific statement that runs for longer than one minute, but do not cease statement execution.

```
CREATE THRESHOLD TH1
  FOR STATEMENT
  TEXT 'SELECT DISTINCT PARTS_BIN FROM STOCK WHERE PART_NUMBER = ?'
  ACTIVITIES ENFORCEMENT DATABASE
  WHEN ACTIVITYTOTALTIME > 1 MINUTE
  COLLECT ACTIVITY DATA WITH DETAILS, SECTION AND VALUES
  CONTINUE
```

CREATE TRANSFORM

The CREATE TRANSFORM statement defines transformation functions, identified by a group name, that are used to exchange structured type values with host language programs and with external functions.

Invocation

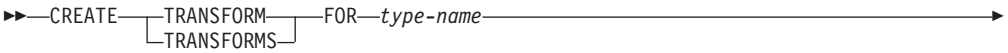
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

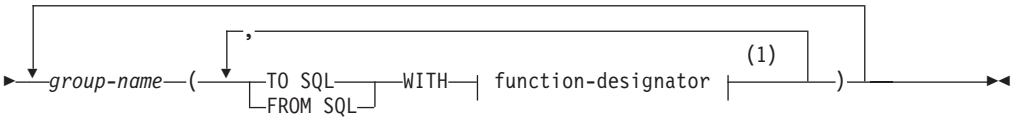
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

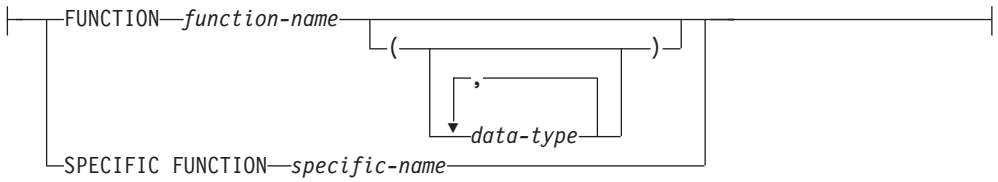
- Owner of the type identified by *type-name*, and EXECUTE privilege on every specified function
- DBADM authority

Syntax





function-designator:



Notes:

1 The same clause must not be specified more than once.

Description

TRANSFORM or TRANSFORMS
Indicates that one or more transform groups is being defined. Either version of the keyword can be specified.

FOR *type-name*
Specifies a name for the user-defined structured type for which the transform group is being defined.

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified *type-name*. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for an unqualified *type-name*. The *type-name* must be the name of an existing user-defined type (SQLSTATE 42704), and it must be a structured type (SQLSTATE 42809). The structured type or any other structured type in the same type hierarchy must not have transforms already defined with the given group-name (SQLSTATE 42739).

group-name
Names the transform group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *group-name* must not identify a transform group that already exists in the catalog for the specified *type-name* (SQLSTATE 42739). The *group-name* must not begin with the characters 'SYS' (SQLSTATE 42939). At most, one of each of the FROM SQL and TO SQL function designations can be specified for any given group (SQLSTATE 42628).

TO SQL
Defines the specific function used to transform a value to the SQL user-defined structured type format. The function must have all its parameters as built-in data types and the returned type is *type-name*.

FROM SQL
Defines the specific function used to transform a value to a built in data type value representing the SQL user-defined structured type. The function must have one parameter of data type *type-name*, and return a built-in data type (or set of built-in data types).

WITH *function-designator*
Uniquely identifies the transform function.

If FROM SQL is specified, *function-designator* must identify a function that meets the following requirements:

- There is one parameter of type *type-name*.
- The return type is a built-in type, or a row whose columns all have built-in types.
- The signature specifies either LANGUAGE SQL or the use of another FROM SQL transform function that has LANGUAGE SQL.

If TO SQL is specified, *function-designator* must identify a function that meets the following requirements:

- All parameters have built-in types.
- The return type is *type-name*.
- The signature specifies either LANGUAGE SQL or the use of another TO SQL transform function that has LANGUAGE SQL.

If *function-designator* identifies a function that does not meet these requirements (according to its use as a FROM SQL or a TO SQL transform function), an error is raised (SQLSTATE 428DC).

Methods (even if specified with FUNCTION ACCESS) cannot be specified as transforms through *function-designator*. Instead, only functions that are defined by the CREATE FUNCTION statement can act as transforms (SQLSTATE 42704 or 42883).

For more information, see “Function, method, and procedure designators” on page 20.

Rules

- The one or more built-in types that are returned from the FROM SQL function should directly correspond to the one or more built-in types that are parameters of the TO SQL function. This is a logical consequence of the inverse relationship between these two functions.

Notes

- When a transform group is not specified in an application program (using the TRANSFORM GROUP precompile or bind option for static SQL, or the SET CURRENT DEFAULT TRANSFORM GROUP statement for dynamic SQL), the transform functions in the transform group 'DB2_PROGRAM' are used (if defined) when the application program is retrieving or sending host variables that are based on the user-defined structured type identified by *type-name*. When retrieving a value of data type *type-name*, the FROM SQL transform is invoked to transform the structured type to the built-in data type returned by the transform function. Similarly, when sending a host variable that will be assigned to a value of data type *type-name*, the TO SQL transform is invoked to transform the built-in data type value to the structured type value. If a user-defined transform group is not specified, or a 'DB2_PROGRAM' group is not defined (for the given structured type), an error is raised (SQLSTATE 42741).
- The built-in data type representation for a structured type host variable must be assignable:
  - from the result of the FROM SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command (using retrieval assignment rules) and
  - to the parameter of the TO SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command (using storage assignment rules).

  If a host variable is not assignment compatible with the type required by the applicable transform function, an error is raised (for bind-in: SQLSTATE 42821; for bind-out: SQLSTATE 42806). For errors that result from string assignments, see “String Assignments”.
- The transform functions identified in the default transform group named 'DB2_FUNCTION' are used whenever a user-defined function not written in SQL is invoked using the data type *type-name* as a parameter or returns type. This applies when the function does not specify the TRANSFORM GROUP clause. When invoking the function with an argument of data type *type-name*, the FROM SQL transform is executed to transform the structured type to the built-in data type returned by the transform function. Similarly, when the returns data type of the function is of data type *type-name*, the TO SQL transform is invoked to transform the built-in data type value returned from the external function program into the structured type value.
- If a structured type contains an attribute that is also a structured type, the associated transform functions must recursively expand (or assemble) all nested structured types. This means that the results or parameters of the transform functions consist only of the set of built-in types representing all base attributes of the subject structured type (including all its nested structured types). There is no "cascading" of transform functions for handling nested structured types.
- The functions identified in this statement are resolved according to the rules outlined previously at the execution of this statement. When these functions are used (implicitly) in subsequent SQL statements, they do not undergo another resolution process. The transform functions defined in this statement are recorded exactly as they are resolved in this statement.

- When attributes or subtypes of a given type are created or dropped, the transform functions for the user-defined structured type must also be changed.
- For a given transform group, the FROM SQL and TO SQL transforms can be specified in either the same *group-name* clause, in separate *group-name* clauses, or in separate CREATE TRANSFORM statements. The only restriction is that a given FROM SQL or TO SQL transform designation may not be redefined without first dropping the existing group definition. This allows you to define, for example, a FROM SQL transform for a given group first, and the corresponding TO SQL transform for the same group at a later time.

Example

Create two transform groups that associate the user-defined structured type polygon with transform functions customized for C and Java, respectively.

```
CREATE TRANSFORM FOR POLYGON
  mystruct1 (FROM SQL WITH FUNCTION myxform_sqlstruct,
             TO SQL WITH FUNCTION myxform_structsql)
  myjava1   (FROM SQL WITH FUNCTION myxform_sqljava,
             TO SQL WITH FUNCTION myxform_javasql)
```

CREATE TRIGGER

The CREATE TRIGGER statement defines a trigger in the database.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- ALTER privilege on the table on which the BEFORE or AFTER trigger is defined
- CONTROL privilege on the view on which the INSTEAD OF trigger is defined
- Owner of the view on which the INSTEAD OF trigger is defined
- ALTERIN privilege on the schema of the table or view on which the trigger is defined
- DBADM authority

and one of:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the trigger does not exist
- CREATEIN privilege on the schema, if the schema name of the trigger refers to an existing schema
- DBADM authority

If the authorization ID of the statement does not have DATAACCESS authority, the privileges (excluding group privileges) held by the authorization ID of the statement must include all of the following authorities, as long as the trigger exists:

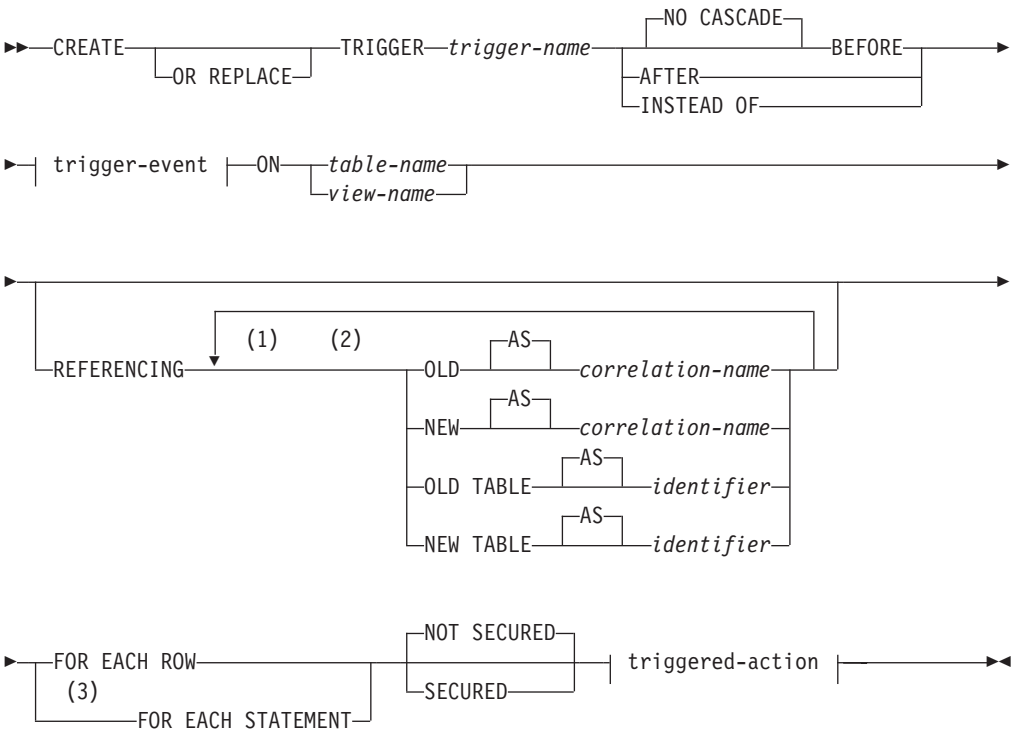
- On the table on which the trigger is defined, if any transition variables or tables are specified:
  - SELECT privilege on the table on which the trigger is defined, if any transition variables or tables are specified
  - CONTROL privilege on the table on which the trigger is defined, if any transition variables or tables are specified
  - DATAACCESS authority
- On any table or view referenced in the triggered action condition:
  - SELECT privilege on any table or view referenced in the triggered action condition
  - CONTROL privilege on any table or view referenced in the triggered action condition
  - DATAACCESS authority
- Necessary privileges to invoke the triggered SQL statements specified.

Group privileges are not considered for any table or view specified in the CREATE TRIGGER statement.

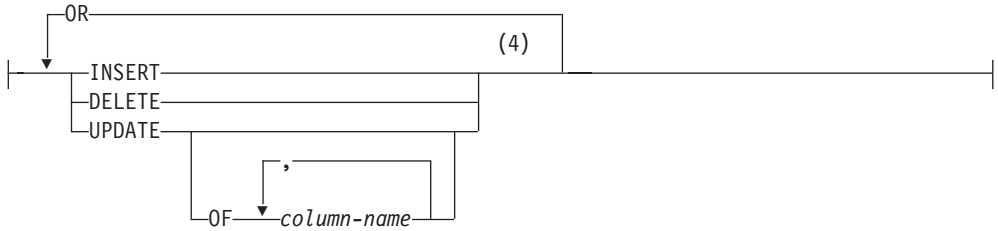
To replace an existing trigger, the authorization ID of the statement must be the owner of the existing trigger (SQLSTATE 42501).

If the SECURED option is specified, the privileges held by the authorization ID of the statement must additionally include SECADM or CREATE_SECURE_OBJECT authority (SQLSTATE 42501).

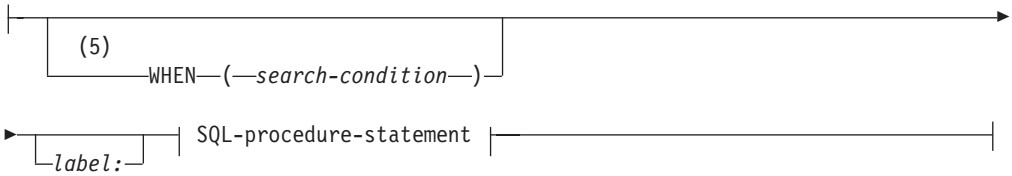
Syntax



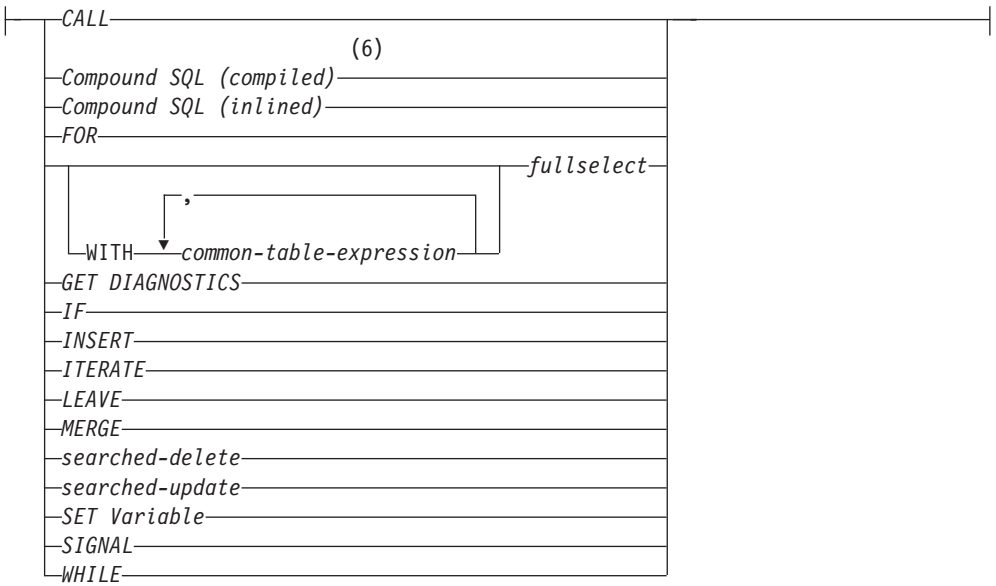
trigger-event:



triggered-action:



SQL-procedure-statement:



Notes:

1 OLD and NEW can only be specified once each.

2 OLD TABLE and NEW TABLE can only be specified once each, and only for AFTER triggers or INSTEAD OF triggers.

3 FOR EACH STATEMENT may not be specified for BEFORE triggers or INSTEAD OF triggers.

4 A trigger event must not be specified more than once for the same operation. For example, INSERT OR DELETE is allowed, but INSERT OR INSERT is not allowed.

5 WHEN condition may not be specified for INSTEAD OF triggers.

6 A compound SQL (compiled) statement cannot be specified if the trigger definition includes a REFERENCING OLD TABLE clause or a REFERENCING NEW TABLE clause. A compound SQL (compiled) statement also cannot be specified for a trigger definition in a partitioned database environment.

Description

OR REPLACE
Specifies to replace the definition for the trigger if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog. This option is ignored if a definition for the trigger does not exist at the current server. This option can be specified only by the owner of the object.

trigger-name
Names the trigger. The name, including the implicit or explicit schema name, must not identify a trigger already described in the catalog (SQLSTATE 42710). If a two-part name is specified, the schema name cannot begin with 'SYS' (SQLSTATE 42939).

NO CASCADE BEFORE
Specifies that the associated triggered action is to be applied before any

changes caused by the actual update of the subject table are applied to the database. It also specifies that the triggered action of the trigger will not cause other triggers to be activated.

AFTER
Specifies that the associated triggered action is to be applied after the changes caused by the actual update of the subject table are applied to the database.

INSTEAD OF
Specifies that the associated triggered action replaces the action against the subject view. Only one INSTEAD OF trigger is allowed for each kind of operation on a given subject view (SQLSTATE 428FP).

trigger-event
Specifies that the triggered action associated with the trigger is to be executed whenever one of the events is applied to the subject table or subject view. Any combination of the events can be specified, but each event (INSERT, DELETE, and UPDATE) can only be specified once (SQLSTATE.42613). If multiple events are specified, the triggered action must be a compound SQL (compiled) statement (SQLSTATE 42601).

INSERT
Specifies that the triggered action associated with the trigger is to be executed whenever an INSERT operation is applied to the subject table or subject view.

DELETE
Specifies that the triggered action associated with the trigger is to be executed whenever a DELETE operation is applied to the subject table or subject view.

UPDATE
Specifies that the triggered action associated with the trigger is to be executed whenever an UPDATE operation is applied to the subject table or subject view, subject to the columns specified or implied.

If the optional *column-name* list is not specified, every column of the table or view is implied. Therefore, omission of the *column-name* list implies that the trigger will be activated by the update of any column of the table or view.

OF *column-name*,...
Each *column-name* specified must be a column of the base table (SQLSTATE 42703). If the trigger is a BEFORE trigger, the *column-name* specified cannot be a generated column other than the identity column (SQLSTATE 42989). No *column-name* can appear more than once in the *column-name* list (SQLSTATE 42711). The trigger will only be activated by the update of a column that is identified in the *column-name* list. This clause cannot be specified for an INSTEAD OF trigger (SQLSTATE 42613).

ON

table-name
Designates the subject table of the BEFORE trigger or AFTER trigger definition. The name must specify a base table or an alias that resolves to a base table (SQLSTATE 42704 or 42809). The name must not specify a catalog table (SQLSTATE 42832), a materialized query table (SQLSTATE 42997), a created temporary table, a declared temporary table (SQLSTATE 42995), or a nickname (SQLSTATE 42809).

view-name
Designates the subject view of the INSTEAD OF trigger definition. The name must specify an untyped view or an alias that resolves to an untyped view with no columns of type XML (SQLSTATE 42704 or 42809). The name must not specify a catalog view (SQLSTATE 42832). The name must not specify a view that is defined using WITH CHECK OPTION (a symmetric view), or a view on which a symmetric view has been defined, directly or indirectly (SQLSTATE 428FQ).

NOT SECURED or SECURED
Specifies whether the trigger is considered secure. The default is NOT SECURED.

NOT SECURED
Specifies the trigger is considered not secure.

SECURED
Specifies the trigger is considered secure. SECURED must be specified for a trigger whose subject table is a table on which row level or column level access control has been activated (SQLSTATE 428H8). Similarly, SECURED must be specified for a trigger that is created on a view and one or more of the underlying tables in that view definition has row level or column level access control activated (SQLSTATE 428H8).

REFERENCING
Specifies the correlation names for the *transition variables* and the table names for the *transition tables*. Correlation names identify a specific row in the set of rows affected by the triggering SQL operation. Table names identify the complete set of affected rows. Each row affected by the triggering SQL operation is available to the triggered action by qualifying columns with *correlation-names* specified as follows.

OLD AS *correlation-name*
Specifies a correlation name which identifies the row state before the triggering SQL operation.

NEW AS *correlation-name*
Specifies a correlation name which identifies the row state as modified by the triggering SQL operation and by any SET statement in a BEFORE trigger that has already executed.

The complete set of rows affected by the triggering SQL operation is available to the triggered action by using a temporary table name specified as follows.

OLD TABLE AS *identifier*
Specifies a temporary table name which identifies the set of affected rows before the triggering SQL operation. If the trigger event is INSERT, the temporary table is empty.

NEW TABLE AS *identifier*
Specifies a temporary table name which identifies the affected rows as modified by the triggering SQL operation and by any SET statement in a BEFORE trigger that has already executed. If the trigger event is DELETE, the temporary table is empty.

The following rules apply to the REFERENCING clause:

- None of the OLD and NEW correlation names and the OLD TABLE and NEW TABLE names can be identical (SQLSTATE 42712).

- Only one OLD and one NEW *correlation-name* may be specified for a trigger (SQLSTATE 42613).
- Only one OLD TABLE and one NEW TABLE *identifier* may be specified for a trigger (SQLSTATE 42613).
- OLD TABLE or NEW TABLE identifiers cannot be defined in a BEFORE trigger (SQLSTATE 42898).
- A NEW transition variable can only be the target of an assignment in a BEFORE trigger. Otherwise, transition variables cannot be the target of an assignment (SQLSTATE 42703 or 42987).
- OLD or NEW correlation names cannot be defined in a FOR EACH STATEMENT trigger (SQLSTATE 42899).
- Transition tables cannot be modified (SQLSTATE 42807).
- The total of the references to the transition table columns and transition variables in the triggered-action cannot exceed the limit for the number of columns in a table or the sum of their lengths cannot exceed the maximum length of a row in a table (SQLSTATE 54040).
- The scope of each *correlation-name* and each *identifier* is the entire trigger definition.
- If the triggered-action includes a compound SQL (compiled) statement:
  - OLD TABLE or NEW TABLE identifiers cannot be defined.
  - If the operation is a DELETE operation, OLD *correlation-name* captures the value of the deleted row. If it is an UPDATE operation, it captures the value of the row before the UPDATE operation. For an insert operation, OLD *correlation-name* captures null values for each column of a row.
  - For an insert operation or an update operation, the value of NEW captures the new state of the row as provided by the original operation and as modified by any BEFORE trigger that has executed to this point. For a delete operation, NEW *correlation-name* captures null values for each column of a row. In a BEFORE DELETE trigger, any non-null values assigned to the new transition variables persist only within the trigger where the assignment occurred.
- If the triggered-action does not include a compound SQL (compiled) statement:
  - The OLD *correlation-name* and the OLD TABLE *identifier* can only be used if the trigger event is either a DELETE operation or an UPDATE operation (SQLSTATE 42898). If the operation is a DELETE operation, OLD *correlation-name* captures the value of the deleted row. If it is an UPDATE operation, it captures the value of the row before the UPDATE operation. The same applies to the OLD TABLE *identifier* and the set of affected rows.
  - The NEW *correlation-name* and the NEW TABLE *identifier* can only be used if the trigger event is either an INSERT operation or an UPDATE operation (SQLSTATE 42898). In both operations, the value of NEW captures the new state of the row as provided by the original operation and as modified by any BEFORE trigger that has executed to this point. The same applies to the NEW TABLE *identifier* and the set of affected rows.

FOR EACH ROW

Specifies that the triggered action is to be applied once for each row of the subject table or subject view that is affected by the triggering SQL operation.

FOR EACH STATEMENT
Specifies that the triggered action is to be applied only once for the whole statement. This type of trigger granularity cannot be specified for a BEFORE trigger or an INSTEAD OF trigger (SQLSTATE 42613). If specified, an UPDATE or DELETE trigger is activated, even if no rows are affected by the triggering UPDATE or DELETE statement.

triggered-action
Specifies the action to be performed when a trigger is activated. A triggered action is composed of an *SQL-procedure-statement* and by an optional condition for the execution of the *SQL-procedure-statement*.

Trigger event predicates can be used anywhere in the triggered action of a CREATE TRIGGER statement that uses a compound SQL (compiled) statement as the *SQL-procedure-statement*.

WHEN

(*search-condition***)**
Specifies a condition that is true, false, or unknown. The *search-condition* provides a capability to determine whether or not a certain triggered action should be executed. The associated action is performed only if the specified search condition evaluates as true. If the WHEN clause is omitted, the associated *SQL-procedure-statement* is always performed.

The WHEN clause cannot be specified for INSTEAD OF triggers (SQLSTATE 42613).

A reference to a transition variable with an XML data type can be used only in a VALIDATED predicate.

label:
Specifies the label for an SQL procedure statement. The label must be unique within a list of SQL procedure statements, including any compound statements nested within the list. Note that compound statements that are not nested can use the same label. A list of SQL procedure statements is possible in a number of SQL control statements.

Only the FOR statement, WHILE statement, and the compound SQL statement can include a label.

SQL-procedure-statement
Specifies the SQL statement that is to be part of the triggered action. A searched update, searched delete, insert, or merge operation on nicknames inside compound SQL is not supported.

The triggered action of a BEFORE trigger on a column of type XML can invoke the XMLVALIDATE function through a SET statement, leave values of type XML unchanged, or assign them to NULL using a SET statement.

The *SQL-procedure-statement* must not contain a statement that is not supported (SQLSTATE 42987).

The *SQL-procedure-statement* cannot reference an undefined transition variable (SQLSTATE 42703), a federated object (SQLSTATE 42997), or a declared temporary table (SQLSTATE 42995). or the start and end columns of the BUSINESS_TIME period (SQLSTATE 42808).

The *SQL-procedure-statement* in a BEFORE trigger cannot:

- Contain any INSERT, DELETE, or UPDATE operations, nor invoke any routine defined with MODIFIES SQL DATA, if it is not a compound SQL (compiled).
- Contain any DELETE or UPDATE operations on the trigger subject table, nor invoke any routine containing such operations, if it is a compound SQL (compiled).
- Reference a materialized query table defined with REFRESH IMMEDIATE (SQLSTATE 42997)
- Reference a generated column other than the identity column in the NEW transition variable (SQLSTATE 42989).

Notes

- Adding a trigger to a table that already has rows in it will not cause any triggered actions to be activated. Thus, if the trigger is designed to enforce constraints on the data in the table, those constraints may not be satisfied by the existing rows.
- If the events for two triggers occur simultaneously (for example, if they have the same event, activation time, and subject tables), then the first trigger created is the first to execute. If the OR REPLACE option is used to replace a previously created trigger, the create time is changed and therefore could affect the order of trigger execution.
- If a column is added to the subject table after triggers have been defined, the following rules apply:
  - If the trigger is an UPDATE trigger that was specified without an explicit column list, then an update to the new column will cause the activation of the trigger.
  - The column will not be visible in the triggered action of any previously defined trigger.
  - The OLD TABLE and NEW TABLE transition tables will not contain this column. Thus, the result of performing a "SELECT *" on a transition table will not contain the added column.
- If a column is added to any table referenced in a triggered action, the new column will not be visible to the triggered action.
- If an object referenced in the trigger body does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the trigger will still be created successfully. The trigger will be marked invalid and will be revalidated the next time it is invoked.
- The result of a fullselect specified in a *SQL-procedure-statement* is not available inside or outside of the trigger.
- A procedure called within a triggered compound statement must not issue a COMMIT or a ROLLBACK statement (SQLSTATE 42985).
- A procedure that contains a reference to a nickname in a searched UPDATE statement, a searched DELETE statement, or an INSERT statement is not supported (SQLSTATE 25000).
- ***Table access restrictions:***: If a procedure is defined as READS SQL DATA or MODIFIES SQL DATA, no statement in the procedure can access a table that is being modified by the compound statement that invoked the procedure (SQLSTATE 57053). If the procedure is defined as MODIFIES SQL DATA, no statement in the procedure can modify a table that is being read or modified by the compound statement that invoked the procedure (SQLSTATE 57053).

- A BEFORE DELETE trigger defined on a table involved in a cycle of cascaded referential constraints should not include references to the table on which it is defined or any other table modified by cascading during the evaluation of the cycle of referential integrity constraints. The results of such a trigger are data dependent and therefore may not produce consistent results.

  In its simplest form, this means that a BEFORE DELETE trigger on a table with a self-referencing referential constraint and a delete rule of CASCADE should not include any references to the table in the *triggered-action*.
- The creation of a trigger causes certain packages to be marked invalid:
  - If an UPDATE trigger without an explicit column list is created, then packages with an update usage on the target table or view are invalidated.
  - If an UPDATE trigger with a column list is created, then packages with update usage on the target table are only invalidated if the package also has an update usage on at least one column in the *column-name* list of the CREATE TRIGGER statement.
  - If an INSERT trigger is created, packages that have an insert usage on the target table or view are invalidated.
  - If a delete trigger is created, packages that have a delete usage on the target table or view are invalidated.
- A package remains invalid until the application program is explicitly bound or rebound, or it is executed and the database manager automatically rebinds it.
- ***Inoperative triggers***: An *inoperative trigger* is a trigger that is no longer available and is therefore never activated. A trigger becomes inoperative if:
  - a privilege that the creator of the trigger is required to have for the trigger to execute is revoked
  - an object such as a table, view or alias, upon which the triggered action is dependent, is dropped
  - a view, upon which the triggered action is dependent, becomes inoperative
  - an alias that is the subject table of the trigger is dropped.

  In practical terms, an inoperative trigger is one in which a trigger definition has been dropped as a result of cascading rules for DROP or REVOKE statements. For example, when a view is dropped, any trigger with an *SQL-procedure-statement* that contains a reference to that view is made inoperative.

  When a trigger is made inoperative, all packages with statements performing operations that were activating the trigger will be marked invalid. When the package is rebound (explicitly or implicitly) the inoperative trigger is completely ignored. Similarly, applications with dynamic SQL statements performing operations that were activating the trigger will also completely ignore any inoperative triggers.

  The trigger name can still be specified in the DROP TRIGGER and COMMENT ON TRIGGER statements.

  An inoperative trigger may be re-created by issuing a CREATE TRIGGER statement using the definition text of the inoperative trigger. This trigger definition text is stored in the TEXT column of the SYSCAT.TRIGGERS catalog view. Note that there is no need to explicitly drop the inoperative trigger in order to re-create it. Issuing a CREATE TRIGGER statement with the same *trigger-name* as an inoperative trigger will cause that inoperative trigger to be replaced with a warning (SQLSTATE 01595).

  Inoperative triggers are indicated by an X in the VALID column of the SYSCAT.TRIGGERS catalog view.

- ***Errors executing triggers***: Errors that occur during the execution of triggered SQL statements are returned using SQLSTATE 09000 unless the error is considered severe. If the error is severe, the severe error SQLSTATE is returned. The SQLERRMC field of the SQLCA for non-severe error will include the trigger name, SQLCODE, SQLSTATE and as many tokens as will fit from the tokens of the failure.

  The *SQL-procedure-statement* could include a SIGNAL SQLSTATE statement or a RAISE_ERROR function. In both these cases, the SQLSTATE returned is the one specified in the SIGNAL SQLSTATE statement or the RAISE_ERROR condition.
- Creating a trigger with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- ***DB2SECURITYLABEL column***: A DB2SECURITYLABEL column can be referenced in the trigger body of a BEFORE TRIGGER but it cannot be changed in the body of a BEFORE trigger (SQLSTATE 42989).
- ***BUSINESS_TIME period columns***: The start and end columns of a BUSINESS_TIME period cannot be changed in the body of BEFORE UPDATE trigger (SQLSTATE 42808).
- ***Read-only views***: The addition of an INSTEAD OF trigger for a view affects the read only characteristic of the view. If a read-only view has a dependency relationship with an INSTEAD OF trigger, the type of operation that is defined for the INSTEAD OF trigger defines whether the view is deletable, insertable, or updatable.
- ***Transition variable values and INSTEAD OF triggers***: The initial values for new transition variables or new transition table columns that are visible in an INSTEAD OF INSERT trigger are set as follows:
  - If a value is explicitly specified for a column in the insert operation, the corresponding new transition variable is that explicitly specified value.
  - If a value is not explicitly specified for a column in the insert operation or the DEFAULT clause is specified, the corresponding new transition variable is:
    - the default value of the underlying table column if the view column is updatable (without the INSTEAD OF trigger)
    - otherwise, the null value

  The initial values for new transition variables that are visible in an INSTEAD OF UPDATE trigger are set as follows:
  - If a value is explicitly specified for a column in the update operation, the corresponding new transition variable is that explicitly specified value.
  - If the DEFAULT clause is explicitly specified for a column in the update operation, the corresponding new transition variable is:
    - the default value of the underlying table column if the view column is updatable (without the INSTEAD OF trigger)
    - otherwise, the null value
  - Otherwise, the corresponding new transition variable is the existing value of the column in the row.
- ***Triggers and typed tables***: A BEFORE or AFTER trigger can be attached to a typed table at any level of a table hierarchy. If an SQL statement activates multiple triggers, the triggers will be executed in their creation order, even if they are attached to different tables in the typed table hierarchy.

When a trigger is activated, its transition variables (OLD, NEW, OLD TABLE and NEW TABLE) may contain rows of subtables. However, they will contain only columns defined on the table to which they are attached.

Effects of INSERT, UPDATE, and DELETE statements:

– Row triggers: When an SQL statement is used to INSERT, UPDATE, or DELETE a table row, it activates row-triggers attached to the most specific table containing the row, and all supertables of that table. This rule is always true, regardless of how the SQL statement accesses the table. For example, when issuing an UPDATE EMP command, some of the updated rows may be in the subtable MGR. For EMP rows, the row-triggers attached to EMP and its supertables are activated. For MGR rows, the row-triggers attached to MGR and its supertables are activated.
– Statement triggers: An INSERT, UPDATE, or DELETE statement activates statement-triggers attached to tables (and their supertables) that could be affected by the statement. This rule is always true, regardless of whether any actual rows in these tables were affected. For example, on an INSERT INTO EMP command, statement-triggers for EMP and its supertables are activated. As another example, on either an UPDATE EMP or DELETE EMP command, statement triggers for EMP and its supertables and subtables are activated, even if no subtable rows were updated or deleted. Likewise, a UPDATE ONLY (EMP) or DELETE ONLY (EMP) command will activate statement-triggers for EMP and its supertables, but not statement-triggers for subtables.

Effects of DROP TABLE statements: A DROP TABLE statement does not activate any triggers that are attached to the table being dropped. However, if the dropped table is a subtable, all the rows of the dropped table are considered to be deleted from its supertables. Therefore, for a table T:

– Row triggers: DROP TABLE T activates row-type delete-triggers that are attached to all supertables of T, for each row of T.
– Statement triggers: DROP TABLE T activates statement-type delete-triggers that are attached to all supertables of T, regardless of whether T contains any rows.

Actions on Views: To predict what triggers are activated by an action on a view, use the view definition to translate that action into an action on base tables. For example:

1. An SQL statement performs UPDATE V1, where V1 is a typed view with a subview V2. Suppose V1 has underlying table T1, and V2 has underlying table T2. The statement could potentially affect rows in T1, T2, and their subtables, so statement triggers are activated for T1 and T2 and all their subtables and supertables.
2. An SQL statement performs UPDATE V1, where V1 is a typed view with a subview V2. Suppose V1 is defined as SELECT ... FROM ONLY(T1) and V2 is defined as SELECT ... FROM ONLY(T2). Since the statement cannot affect rows in subtables of T1 and T2, statement triggers are activated for T1 and T2 and their supertables, but not their subtables.
3. An SQL statement performs UPDATE ONLY(V1), where V1 is a typed view defined as SELECT ... FROM T1. The statement can potentially affect T1 and its subtables. Therefore, statement triggers are activated for T1 and all its subtables and supertables.
4. An SQL statement performs UPDATE ONLY(V1), where V1 is a typed view defined as SELECT ... FROM ONLY(T1). In this case, T1 is the only table that

can be affected by the statement, even if V1 has subviews and T1 has subtables. Therefore, statement triggers are activated only for T1 and its supertables.

- ***MERGE statement and triggers***: The MERGE statement can execute update, delete, and insert operations. The applicable UPDATE, DELETE, or INSERT triggers are activated for the MERGE statement when an update, delete, or insert operation is executed.
- ***Obfuscation***: The CREATE TRIGGER statement can be submitted in obfuscated form. In an obfuscated statement, only the trigger name is readable. The rest of the statement is encoded in such a way that is not readable but can be decoded by the database server. Obfuscated statements can be produced by calling the DBMS_DDL.WRAP function.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - OLD_TABLE can be specified in place of OLD TABLE, and NEW_TABLE can be specified in place of NEW TABLE
  - MODE DB2SQL can be specified following FOR EACH ROW or FOR EACH STATEMENT
- ***Creating a trigger with the SECURED option***: Normally users with SECADM authority do not have privileges to create database objects such as triggers or user-defined functions. Typically, they will examine the data accessed by a trigger, ensure it is secure, then grant the CREATE_SECURE_OBJECT authority to someone who has the required privileges to create the secure trigger. After the trigger is created, they will revoke the CREATE_SECURE_OBJECT authority from the trigger owner.

  The trigger is considered secure. The database manager treats the SECURED attribute as an assertion that declares the user has established an audit procedure for all activities in the trigger body. If a secure trigger references user-defined functions, the database manager assumes those functions are secure without validation. If those functions can access sensitive data, the user with SECADM authority needs to ensure those functions are allowed to access those data and that all subsequent ALTER FUNCTION statements or changes to external packages are being reviewed by this audit process.

  A trigger must be secure if its subject table has row level or column level access control activated. Similarly, a trigger must be secure if its subject table is a view and one or more of the underlying tables in the view definition has row level or column level access control activated.
- ***Creating a trigger with the NOT SECURED option***: The CREATE TRIGGER statement returns an error if the trigger's subject table has row level or column level access control activated. Similarly, the CREATE TRIGGER statement fails if the trigger is defined on a view and one or more of the underlying tables in that view definition has row level or column level access control activated.
- ***Row and column access control that is not enforced for transition variables and transition tables***: Triggers are used for database integrity, and as such a balance between security and database integrity is needed. If row level or column level access control is activated on the subject table or an underlying table of the subject view, row permissions and column masks are not applied to the initial values of transition variables and transition tables. Row level and column level access control that is enforced for the subject table or an underlying table of the subject view is also ignored for transition variables and transition tables that are referenced in the trigger body or are passed as arguments to user-defined functions invoked in the trigger body. To ensure there is no security concern for

SQL statements in the trigger action to access sensitive data in transition variables and transition tables, the trigger must be created with the SECURED option. If a trigger is not secure, the CREATE TRIGGER statement returns an error.

- ***Considerations for implicitly hidden columns***: A transition variable exists for any column defined as implicitly hidden. In the body of a trigger, a transition variable that corresponds to an implicitly hidden column can be referenced.

Examples

- *Example 1:* Create two triggers that will result in the automatic tracking of the number of employees a company manages. The triggers will interact with the following tables:
  - EMPLOYEE table with these columns: ID, NAME, ADDRESS, and POSITION.
  - COMPANY_STATS table with these columns: NBEMP, NBPRODUCT, and REVENUE.

  The first trigger increments the number of employees each time a new person is hired; that is, each time a new row is inserted into the EMPLOYEE table:

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

  The second trigger decrements the number of employees each time an employee leaves the company; that is, each time a row is deleted from the table EMPLOYEE:

```
CREATE TRIGGER FORMER_EMP
  AFTER DELETE ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP - 1
```

- *Example 2:* Create a trigger that ensures that whenever a parts record is updated, the following check and (if necessary) action is taken:
  - If the on-hand quantity is less than 10% of the maximum stocked quantity, then issue a shipping request ordering the number of items for the affected part to be equal to the maximum stocked quantity minus the on-hand quantity.

  The trigger will interact with the PARTS table with these columns: PARTNO, DESCRIPTION, ON_HAND, MAX_STOCKED, and PRICE.

  ISSUE_SHIP_REQUEST is a user-defined function that sends an order form for additional parts to the appropriate company.

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (N.ON_HAND < 0.10 * N.MAX_STOCKED)
  BEGIN ATOMIC
  VALUES(ISSUE_SHIP_REQUEST(N.MAX_STOCKED - N.ON_HAND, N.PARTNO));
  END
```

- *Example 3:* Repeat the scenario in Example 2 except use a fullselect instead of a VALUES statement to invoke the user-defined function. This example also shows how to define the trigger as a statement trigger instead of a row trigger. For each row in the transition table that evaluates to true for the WHERE clause, a shipping request is issued for the part.

```
CREATE TRIGGER REORDER
   AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
   REFERENCING NEW TABLE AS NTABLE
   FOR EACH STATEMENT
```

```
      BEGIN ATOMIC
         SELECT ISSUE_SHIP_REQUEST(MAX_STOCKED - ON_HAND, PARTNO)
            FROM NTABLE
         WHERE (ON_HAND < 0.10 * MAX_STOCKED);
   END
```

- *Example 4:* Create a trigger that will cause an error when an update occurs that would result in a salary increase greater than ten percent of the current salary.

```
CREATE TRIGGER RAISE_LIMIT
  AFTER UPDATE OF SALARY ON EMPLOYEE
  REFERENCING NEW AS N OLD AS O
  FOR EACH ROW
  WHEN (N.SALARY > 1.1 * O.SALARY)
        SIGNAL SQLSTATE '75000' SET MESSAGE_TEXT='Salary increase>10%'
```

- *Example 5:* Consider an application which records and tracks changes to stock prices. The database contains two tables, CURRENTQUOTE and QUOTEHISTORY.

```
Tables: CURRENTQUOTE (SYMBOL, QUOTE, STATUS)
        QUOTEHISTORY (SYMBOL, QUOTE, QUOTE_TIMESTAMP)
```

  When the QUOTE column of CURRENTQUOTE is updated, the new quote should be copied, with a timestamp, to the QUOTEHISTORY table. Also, the STATUS column of CURRENTQUOTE should be updated to reflect whether the stock is:

  1. rising in value;
  2. at a new high for the year;
  3. dropping in value;
  4. at a new low for the year;
  5. steady in value.

  CREATE TRIGGER statements that accomplish this are as follows.

  – Trigger Definition to set the status:

```
CREATE TRIGGER STOCK_STATUS
  NO CASCADE BEFORE UPDATE OF QUOTE ON CURRENTQUOTE
  REFERENCING NEW AS NEWQUOTE OLD AS OLDQUOTE
  FOR EACH ROW
  BEGIN ATOMIC
     SET NEWQUOTE.STATUS =
       CASE
          WHEN NEWQUOTE.QUOTE >
                (SELECT MAX(QUOTE) FROM QUOTEHISTORY
                WHERE SYMBOL = NEWQUOTE.SYMBOL
                AND YEAR(QUOTE_TIMESTAMP) = YEAR(CURRENT DATE) )
             THEN 'High'
          WHEN NEWQUOTE.QUOTE <
(SELECT MIN(QUOTE) FROM QUOTEHISTORY
                WHERE SYMBOL = NEWQUOTE.SYMBOL
                AND YEAR(QUOTE_TIMESTAMP) = YEAR(CURRENT DATE) )
             THEN 'Low'
          WHEN NEWQUOTE.QUOTE > OLDQUOTE.QUOTE
             THEN 'Rising'
          WHEN NEWQUOTE.QUOTE < OLDQUOTE.QUOTE
             THEN 'Dropping'
          WHEN NEWQUOTE.QUOTE = OLDQUOTE.QUOTE
             THEN 'Steady'
       END;
  END
```

  – Trigger Definition to record change in QUOTEHISTORY table:

```
CREATE TRIGGER RECORD_HISTORY
  AFTER UPDATE OF QUOTE ON CURRENTQUOTE
  REFERENCING NEW AS NEWQUOTE
```

```
FOR EACH ROW
BEGIN ATOMIC
  INSERT INTO QUOTEHISTORY
    VALUES (NEWQUOTE.SYMBOL, NEWQUOTE.QUOTE, CURRENT TIMESTAMP);
END
```

- *Example 6:*Create a trigger that overrides any changes to the location field in the employee record in the org table. This trigger would be useful if new employee records acquired when a smaller company was purchased are processed and the target location allocated to the employee is 'Toronto' and the new target location is 'Los Angeles'. The before trigger will ensure that regardless what value the application allocates for this field, that the final resultant value is 'Los Angeles'.

```
CREATE TRIGGER LOCATION_TRIGGER
  NO CASCADE
   BEFORE UPDATE ON ORG
   REFERENCING
     OLD AS PRE
     NEW AS POST
   FOR EACH ROW
   WHEN (POST.LOCATION = 'Toronto')
      SET POST.LOCATION = 'Los Angeles';
   END
```

- *Example 7:* Create a BEFORE trigger that automatically validates XML documents containing new product descriptions before they are inserted into the PRODUCT table of the SAMPLE database:

```
CREATE TRIGGER NEWPROD NO CASCADE BEFORE INSERT ON PRODUCT
   REFERENCING NEW AS N
   FOR EACH ROW
   BEGIN ATOMIC
      SET (N.DESCRIPTION) = XMLVALIDATE(N.DESCRIPTION
         ACCORDING TO XMLSCHEMA ID product);
   END
```

CREATE TRUSTED CONTEXT

The CREATE TRUSTED CONTEXT statement defines a trusted context at the current server.

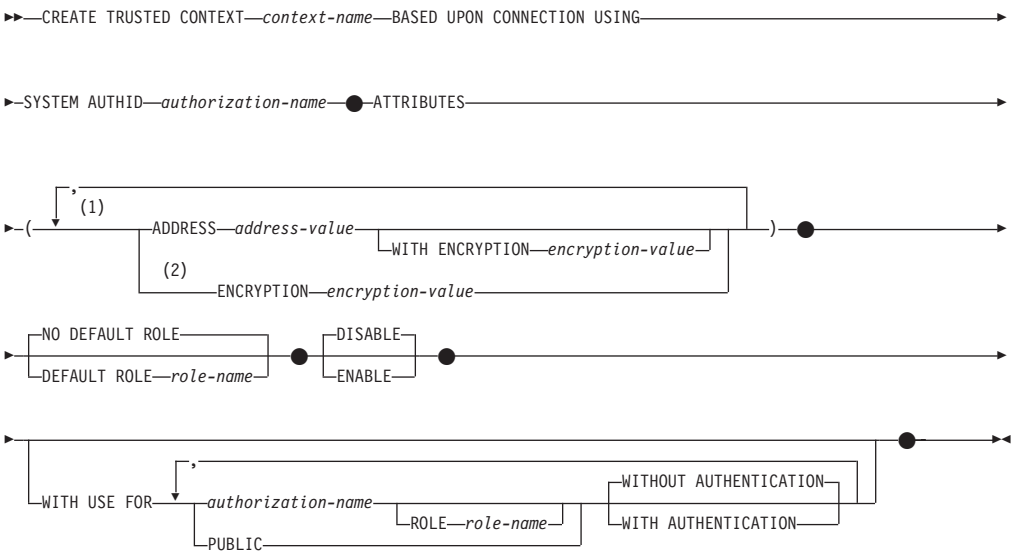
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Notes:

1 Each combination of an attribute name and its corresponding value, as a pair, must be unique (SQLSTATE 4274D).

2 ENCRYPTION cannot be specified more than once (SQLSTATE 42614); however, WITH ENCRYPTION can be specified for each ADDRESS that is specified.

Description

context-name
Names the trusted context. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The name must not identify a trusted context that already exists at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

BASED UPON CONNECTION USING SYSTEM AUTHID *authorization-name*
Specifies that the context is a connection established by system authorization

ID *authorization-name*, which must not be associated with an existing trusted context (SQLSTATE 428GL). It cannot be the authorization ID of the statement (SQLSTATE 42502).

ATTRIBUTES (...)
Specifies a list of one or more connection trust attributes upon which the trusted context is defined.

ADDRESS *address-value*
Specifies the actual communication address used by the client to communicate with the database server. The only protocol supported is TCP/IP. The ADDRESS attribute can be specified multiple times, but each *address-value* pair must be unique for the set of attributes (SQLSTATE 4274D).

When establishing a trusted connection, if multiple values are defined for the ADDRESS attribute of a trusted context, a candidate connection is considered to match this attribute if the address used by the connection matches any of the defined values for the ADDRESS attribute of the trusted context.

address-value
Specifies a string constant that contains the value to be associated with the ADDRESS trust attribute. The *address-value* must be an IPv4 address, an IPv6 address, or a secure domain name.

- An IPv4 address must not contain leading spaces and is represented as a dotted decimal address. An example of an IPv4 address is 9.112.46.111. The value 'localhost' or its equivalent representation '127.0.0.1' will not result in a match; the real IPv4 address of the host must be specified instead.
- An IPv6 address must not contain leading spaces and is represented as a colon hexadecimal address. An example of an IPv6 address is 2001:0DB8:0000:0000:0008:0800:200C:417A. IPv4-mapped IPv6 addresses (for example, ::ffff:192.0.2.128) will not result in a match. Similarly, 'localhost' or its IPv6 short representation '::1' will not result in a match.
- A domain name is converted to an IP address by the domain name server where a resulting IPv4 or IPv6 address is determined. An example of a domain name is corona.torolab.ibm.com. When a domain name is converted to an IP address, the result of this conversion could be a set of one or more IP addresses. In this case, an incoming connection is said to match the ADDRESS attribute of a trusted context object if the IP address from which the connection originates matches any of the IP addresses to which the domain name was converted. When creating a trusted context object, it is advantageous to provide domain name values for the ADDRESS attribute instead of static IP addresses, particularly in Dynamic Host Configuration Protocol (DHCP) environments. With DHCP, a device can have a different IP address each time it connects to the network. So, if a static IP address is provided for the ADDRESS attribute of a trusted context object, some device might acquire a trusted connection unintentionally. Providing domain names for the ADDRESS attribute of a trusted context object avoids this problem in DHCP environments.

WITH ENCRYPTION *encryption-value*
Specifies the minimum level of encryption of the data stream or

network encryption for this specific *address-value*. This *encryption-value* overrides the global ENCRYPTION attribute setting for this specific *address-value*.

encryption-value
Specifies a string constant that contains the value to be associated with the ENCRYPTION trust attribute for this specific *address-value*. The *encryption-value* must be one of the following values (SQLSTATE 42615):

- NONE, no specific level of encryption is required
- LOW, a minimum of light encryption is required; the authentication type on the database manager must be DATA_ENCRYPT if an incoming connection is to match the encryption setting for this specific address
- HIGH, Secure Sockets Layer (SSL) encryption must be used for data communication between the DB2 client and the DB2 server if an incoming connection is to match the encryption setting for this specific address

ENCRYPTION *encryption-value*
Specifies the minimum level of encryption of the data stream or network encryption. The default is NONE.

encryption-value
Specifies a string constant that contains the value to be associated with the ENCRYPTION trust attribute for this specific *address-value*. The *encryption-value* must be one of the following values (SQLSTATE 42615):

- NONE, no specific level of encryption is required for an incoming connection to match the ENCRYPTION attribute of this trusted context object
- LOW, a minimum of light encryption is required; the authentication type on the database manager must be DATA_ENCRYPT if an incoming connection is to match the ENCRYPTION attribute of this trusted context object
- HIGH, Secure Sockets Layer (SSL) encryption must be used for data communication between the DB2 client and the DB2 server if an incoming connection is to match the ENCRYPTION attribute of this trusted context object

The following table summarizes when a trusted context can be used, depending on the encryption used by the existing connection. If the trusted context cannot be used for the connection, a warning is returned (SQLSTATE 01679) and the SQLWARN8 field of the SQLCA is set to 'Y', indicating that the connection is a regular (non-trusted) connection.

Table 27. Encryption and trusted contexts

Encryption used by existing connection	ENCRYPTION value for trusted context	Can the trusted context be used for the connection?
No encryption	'NONE'	Yes
No encryption	'LOW'	No
No encryption	'HIGH'	No

Table 27. Encryption and trusted contexts (continued)

Encryption used by existing connection	ENCRYPTION value for trusted context	Can the trusted context be used for the connection?
Low encryption (DATA_ENCRYPT)	'NONE'	Yes
Low encryption (DATA_ENCRYPT)	'LOW'	Yes
Low encryption (DATA_ENCRYPT)	'HIGH'	No
High encryption (SSL)	'NONE'	Yes
High encryption (SSL)	'LOW'	Yes
High encryption (SSL)	'HIGH'	Yes

NO DEFAULT ROLE or DEFAULT ROLE *role-name*
Specifies whether or not a default role is associated with a trusted connection that is based on this trusted context. The default is NO DEFAULT ROLE.

NO DEFAULT ROLE
Specifies that the trusted context does not have a default role.

DEFAULT ROLE *role-name*
Specifies that *role-name* is the default role for the trusted context. The *role-name* must identify a role that exists at the current server (SQLSTATE 42704). This role is used with the user in a trusted connection, based on this trusted context, when the user does not have a user-specific role defined as part of the definition of the trusted context.

DISABLE or ENABLE
Specifies whether the trusted context is created in the enabled or disabled state. The default is DISABLE.

DISABLE
Specifies that the trusted context is created in the disabled state. A trusted context that is disabled is not considered when a trusted connection is established.

ENABLE
Specifies that the trusted context is created in the enabled state.

WITH USE FOR
Specifies who can use a trusted connection that is based on this trusted context.

authorization-name
Specifies that the trusted connection can be used by the specified *authorization-name*. The *authorization-name* must not be specified more than once in the WITH USE FOR clause (SQLSTATE 428GM). It must also not be the authorization ID of the statement (SQLSTATE 42502). If the definition of a trusted context allows access by both PUBLIC and a list of users, the specifications for a user override the specifications for PUBLIC. For example, assume that a trusted context is defined that allows access by both PUBLIC WITH AUTHENTICATION and JOE WITHOUT AUTHENTICATION. If the trusted context is used by JOE, authentication is not required. However, if the trusted context is used by GEORGE, authentication is required.

ROLE *role-name*
Specifies that *role-name* is the role to be used for the user when a trusted connection is using the trusted context. The *role-name* must identify a role that exists at the current server (SQLSTATE 42704). The role explicitly specified for the user overrides any default role associated with the trusted context.

PUBLIC
Specifies that a trusted connection that is based on this trusted context can be used by any user. PUBLIC must not be specified more than once (SQLSTATE 428GM). All users using such a trusted connection make use of the privileges associated with the default role for the associated trusted context. If a default role is not defined for the trusted context, there is no role associated with the users that use a trusted connection based on this trusted context.

WITHOUT AUTHENTICATION or WITH AUTHENTICATION
Specifies whether or not switching the user on a trusted connection requires authentication of the user. The default is WITHOUT AUTHENTICATION.

WITHOUT AUTHENTICATION
Specifies that switching the current user on a trusted connection to this user does not require authentication.

WITH AUTHENTICATION
Specifies that switching the current user on a trusted connection to this user requires authentication.

Rules

- A trusted context-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). Trusted context-exclusive SQL statements are:
  - CREATE TRUSTED CONTEXT, ALTER TRUSTED CONTEXT, or DROP (TRUSTED CONTEXT)
- A trusted context-exclusive SQL statement cannot be issued within a global transaction; for example, an XA transaction or a global transaction that is initiated as part of two-phase commit for federated transactions (SQLSTATE 51041).

Notes

- When providing an IP address as part of a trusted context definition, the address must be in the format that is in effect for the network. For example, providing an address in an IPv6 format when the network is IPv4 will not result in a match. In a mixed environment, it is advantageous to specify both the IPv4 and the IPv6 representations of the address, or better yet, to specify a secure domain name (for example, corona.torolab.ibm.com), which hides the address format details.
- ***Specifying a role in the definition of a trusted context:*** The definition of a trusted context can designate a role for a specific authorization ID, and a default role to be used for authorization IDs for which a specific role has not been specified in the definition of the trusted context. This role can be used with a trusted connection based on the trusted context, but it does not make the role available outside of a trusted connection based on the trusted context.
- When issuing a data manipulation language (DML) SQL statement using a trusted connection, the privileges held by a context-assigned role in effect for the

authorization ID within the definition of the associated trusted context are considered in addition to other privileges directly held by the authorization ID of the statement, or indirectly by other roles held by the authorization ID of the statement.

- The privileges held by a context-assigned role in effect for the authorization ID within the definition of the associated trusted context are not considered for data definition language (DDL) SQL statements. For example, to create an object, the authorization ID of the statement must be able to do so without including the privileges held by the context-assigned role.
- When installing a new application that authenticates to DB2 using the same credentials as an existing application on the same machine, and which takes advantage of a trusted context, the new application might also take advantage of the same trusted context object (inheriting the trusted context role, for example). This might not be the security administrator's intention. The security administrator might want to turn on the DB2 audit facility to find out what applications are taking advantage of trusted context objects.
- Only one uncommitted trusted context-exclusive SQL statement is allowed at a time across all database partitions. If an uncommitted trusted context-exclusive SQL statement is executing, subsequent trusted context-exclusive SQL statements will wait until the current trusted context-exclusive SQL statement commits or rolls back.
- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.

Examples

- *Example 1:* Create a trusted context such that the current user on a trusted connection based on this trusted context can be switched to two different user IDs. When the current user of the connection is switched to user ID JOE, authentication is not required. However, authentication is required when the current user of the connection is switched to user ID BOB. Note that the trusted context has a default role called *context-role*. This implies that users working within the confines of this trusted context inherit the privileges associated with role *context-role*.

```
CREATE TRUSTED CONTEXT APPSERVER
  BASED UPON CONNECTION USING SYSTEM AUTHID WRJAIBI
  DEFAULT ROLE CONTEXT_ROLE
  ENABLE
  ATTRIBUTES (ADDRESS '9.26.113.204')
  WITH USE FOR JOE WITHOUT AUTHENTICATION
    BOB WITH AUTHENTICATION
```

- *Example 2:* Create a trusted context such that the current user of a trusted connection based on this trusted context can be switched to any user ID without authentication.

```
CREATE TRUSTED CONTEXT SECUREROLE
  BASED UPON CONNECTION USING SYSTEM AUTHID PBIRD
  ENABLE
  ATTRIBUTES (ADDRESS '9.26.113.204')
  WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
```

- *Example 3:* Create a trusted context such that the current user of a trusted connection based on this trusted context can be switched to any user ID without authentication. The difference between this trusted context and the trusted context created in example 2, is that this trusted context has an additional attribute called ENCRYPTION. The ENCRYPTION attribute setting for trusted context SECUREROLEENCRYPT states that the encryption setting used by a connection must be at least "low encryption" (see Table 27 on page 805) to match this trusted context attribute.

```
CREATE TRUSTED CONTEXT SECUREROLEENCRYPT
  BASED UPON CONNECTION USING SYSTEM AUTHID SHARPER
  ENABLE
  ATTRIBUTES (ADDRESS '9.26.113.204'
    ENCRYPTION 'LOW')
  WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
```

- *Example 4:* Create a trusted context, such that connections made by user WRJAIBI from addresses 9.26.146.201 and 9.26.146.203 are trusted when no encryption is used, but a connection made by user WRJAIBI from address 9.26.146.202 requires a LOW level of encryption to be trusted.

```
CREATE TRUSTED CONTEXT WALIDLOCSENSITIVE
  BASED UPON CONNECTION USING SYSTEM AUTHID WRJAIBI
  ENABLE
  ATTRIBUTES (ADDRESS '9.26.146.201',
    ADDRESS '9.26.146.202' WITH ENCRYPTION 'LOW',
    ADDRESS '9.26.146.203'
    ENCRYPTION 'NONE')
```

CREATE TYPE

The CREATE TYPE statement defines a user-defined data type at the current server.

Five different kinds of user-defined data types can be created using this statement. Each of these types is described separately.

- Array. A user-defined data type that is an ordinary array or an associative array. The elements of an array type are based on one of the built-in data types or a user-defined type other than a cursor type or structured type.
- Cursor. A user-defined data type that is a cursor type.
- Distinct. A user-defined data type that is sourced on one of the built-in data types and can be defined to use strong type rules or weak type rules.. Functions that cast between the user-defined distinct type and the source built-in data type are generated when a strongly typed distinct type is created. Optionally, support for comparison operations to use with the strongly typed distinct type can be generated when the user-defined distinct type is created.
- Row. A user-defined data type that represents a row. It includes one or more fields with associated data types that make up a row of data.
- Structured. A user-defined data type that represents an object and associated methods. It may include zero or more attributes and may be a subtype allowing attributes to be inherited from a supertype. Some methods are generated when the user-defined structured type is created and others can be specified as part of the definition.

CREATE TYPE (array)

The CREATE TYPE (array) statement defines an array type. The elements of an array type are based on one of the built-in data types or a user-defined distinct type.

Invocation

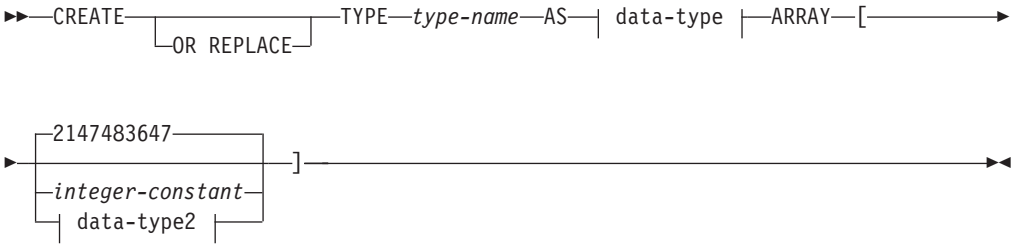
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

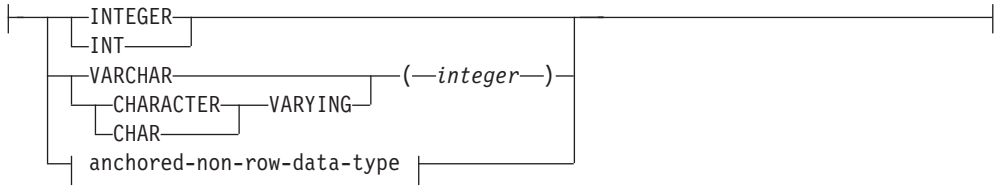
- IMPLICIT_SCHEMA authority on the database, if the schema name of the array type does not refer to an existing schema
- CREATEIN privilege on the schema, if the schema name of the array type refers to an existing schema
- DBADM authority

Syntax



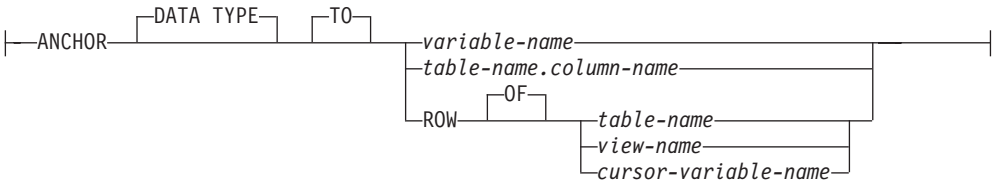
data-type:

data-type2:

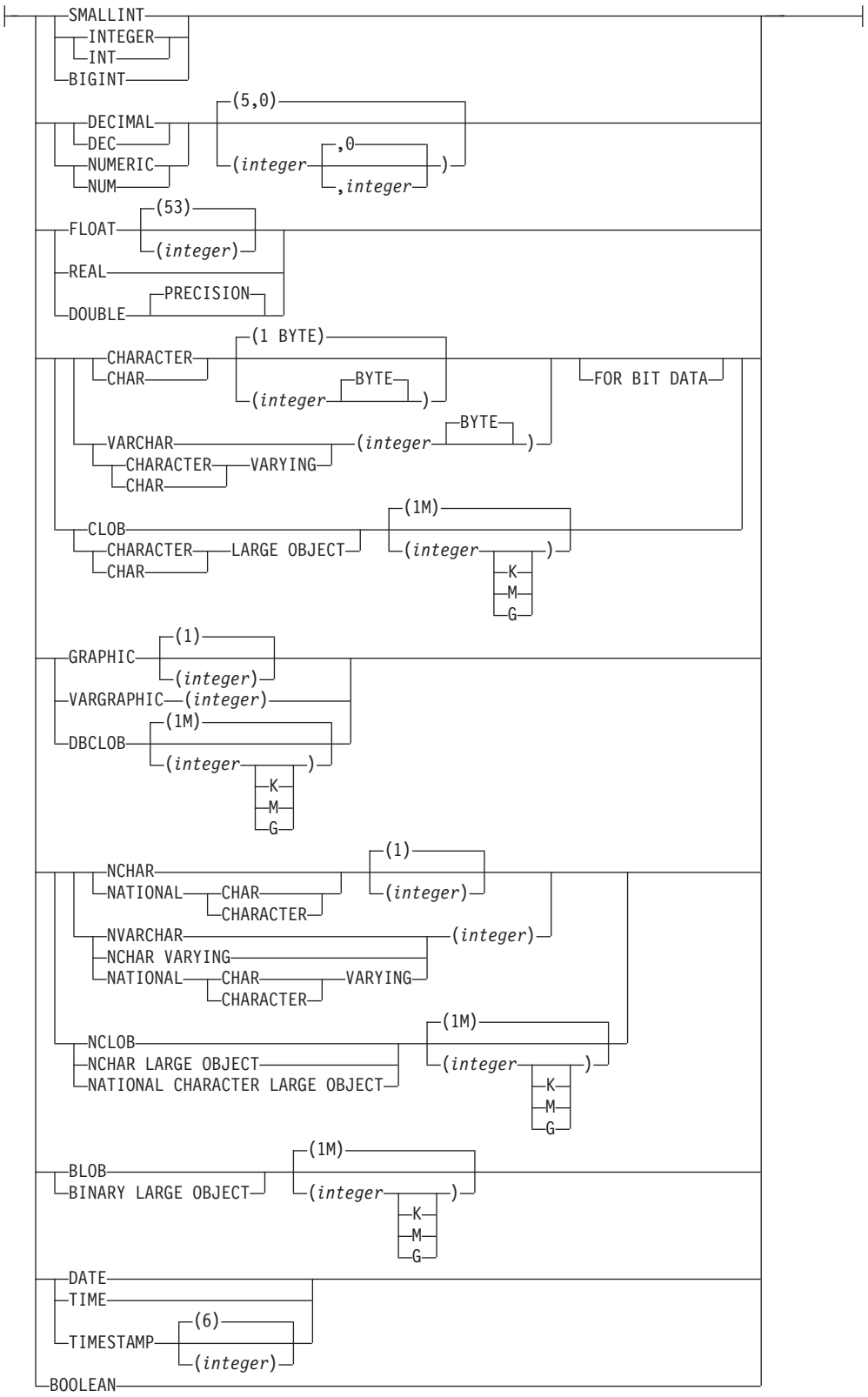


anchored-data-type:

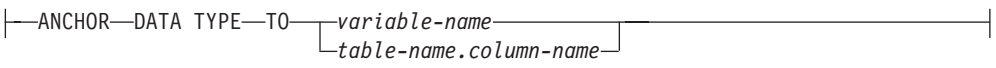
CREATE TYPE (array)



built-in-type:


SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer
,0
,integer)
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
CHARACTER
CHAR
(1 BYTE)
(integer
BYTE)
FOR BIT DATA
VARCHAR
CHARACTER VARYING
CHAR VARYING
(integer
BYTE)
CLOB
CHARACTER LARGE OBJECT
CHAR LARGE OBJECT
(1M)
(integer
K
M
G)
GRAPHIC
(1)
(integer)
VARGRAPHIC(integer)
DBCLOB
(1M)
(integer
K
M
G)
NCHAR
NATIONAL CHAR
NATIONAL CHARACTER
(1)
(integer)
NVARCHAR
NCHAR VARYING
NATIONAL CHAR VARYING
NATIONAL CHARACTER VARYING
(integer)
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
(1M)
(integer
K
M
G)
BLOB
BINARY LARGE OBJECT
(1M)
(integer
K
M
G)
DATE
TIME
TIMESTAMP
(6)
(integer)
BOOLEAN

anchored-non-row-data-type:



Description

OR REPLACE
Specifies to replace the definition for the data type if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that functions and methods are invalidated instead of dropped when they have parameters or a return value defined with the data type being replaced. The existing definition must not be a structured type (SQLSTATE 42809). This option is ignored if a definition for the data type does not exist at the current server.

type-name
Names the type. The name, including the implicit or explicit qualifier, must not identify any other type (built-in or user-defined) that already exists at the current server. The unqualified name must not be the same as the name of a built-in data type or BOOLEAN, BINARY or VARBINARY (SQLSTATE 42918).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *type-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

If a two-part *type-name* is specified, the schema name must not begin with the characters 'SYS' (SQLSTATE 42939).

data-type
Specifies the data type of the array elements.

built-in-type
Specifies a built-in data type. See "CREATE TABLE" for the description of built-in data types. Built-in types include the data types described in “CREATE TABLE”, other than reference, SYSPROC.DB2SECURITYLABEL, XML, or user-defined types (SQLSTATE 429C2).

row-type-name
Specifies the name of a user-defined row type. If a *row-type-name* is specified without a schema name, the *row-type-name* is resolved by searching the schemas in the SQL path. Row types can be nested as elements in other array types with a maximum nesting level of sixteen.

array-type-name
Specifies an array type. If an *array-type-name* is specified without a schema name, the *array-type-name* is resolved by searching the schemas in the SQL path. Array types can be nested as elements in other array types with a maximum nesting level of sixteen.

anchored-data-type
Identifies another object used to determine the data type. The data type of the anchor object is bound by the same limitations that apply when specifying the data type directly, or in the case of a row, to creating a row type.

ANCHOR DATA TYPE TO
Indicates that an anchored data type is used to specify the data type.

variable-name
Identifies a global variable. The data type of the global variable is used as the data type for the array elements.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for the array elements.

ROW OF *table-name* **or** *view-name*
Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*.The data type of the array elements is an unnamed row type.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following elements (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a *select-statement* where all the result columns are named.

If the cursor type of the cursor variable is not strongly-typed using a named row type, the data type of the array elements is an unnamed row type.

anchored-non-row-data-type
Identifies another object used to determine the data type. The data type of the anchor object is bound by the same limitations that apply when specifying the data type directly.

ANCHOR DATA TYPE TO
Indicates that an anchored data type is used to specify the data type.

variable-name
Identifies a global variable with a data type that is an INTEGER or VARCHAR data type. The data type of the global variable is used as the data type for the array index.

table-name.column-name
Identifies a column name of an existing table or view with a data type that is an INTEGER or VARCHAR data type. The data type of the column is used as the data type for the array index.

ARRAY [*integer-constant***]**
Specifies that the type is an array with a maximum cardinality of *integer-constant*. The value must be a positive integer (not zero) and less than the largest positive integer value (SQLSTATE 42820). The default is the largest positive integer value (2 147 483 647). The cardinality of an array value is determined by the highest element position assigned to the array value.

The maximum cardinality of an array on a given system is limited by the total amount of memory available to DB2 applications. As such, although arrays of large cardinalities can be created, not all elements might be available for use.

ARRAY[*data-type2***]**
Specifies that the type is an associative array that is indexed with values of data type *data-type2*. The data type must be either the INTEGER or VARCHAR data type (SQLSTATE 429C2). The values specified as the index when

assigning an array element must be assignable to a value of *data-type2*. The cardinality of an array value is determined by the number of unique index values used when assigning array elements.

Rules

- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

Notes

- ***Array type usage:*** An array type can only be used as the data type of:
  - A local variable in a compound SQL (compiled) statement
  - A parameter of an SQL routine
  - A parameter of a Java procedure (non-nested ordinary arrays only)
  - The returns type of an SQL function
  - A global variable
- A variable or parameter defined with an array type can only be used in compound SQL (compiled) statements

Examples

Example 1: Create an array type named PHONENUMBERS with a maximum of 50 elements that are of the DECIMAL(10, 0) data type.

```
CREATE TYPE PHONENUMBERS AS DECIMAL(10,0)
  ARRAY[50]
```

Example 2: Create an array type named NUMBERS with the default number of elements in the schema GENERIC.

```
CREATE TYPE GENERIC.NUMBERS AS DECFLOAT(34)
  ARRAY[]
```

Example 3: Create an associative array named PERSONAL_PHONENUMBERS with elements that are DECIMAL(16, 0) that is indexed by strings like 'Home', 'Work', or 'Mom'.

```
CREATE TYPE PERSONALPHONENUMBERS AS DECIMAL(16, 0) ARRAY[VARCHAR(8)]
```

Example 4: Create an associative array type where the indexes are province, territory, or country names and the elements are capital cities:

```
CREATE TYPE CAPITALSARRAY AS VARCHAR(30) ARRAY[VARCHAR(20)]
```

Example 5: Create an associative array type for product descriptions of up to 40 characters long, where the indexes are the product numbers, which are a maximum of 12 characters long:

```
CREATE TYPE PRODUCTS AS VARCHAR(40) ARRAY[VARCHAR(12)]
```

CREATE TYPE (cursor)

The CREATE TYPE (cursor) statement defines a user-defined cursor type.

Invocation

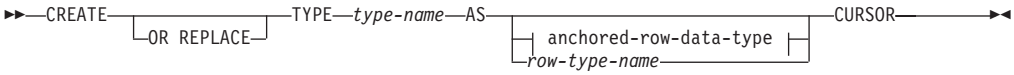
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

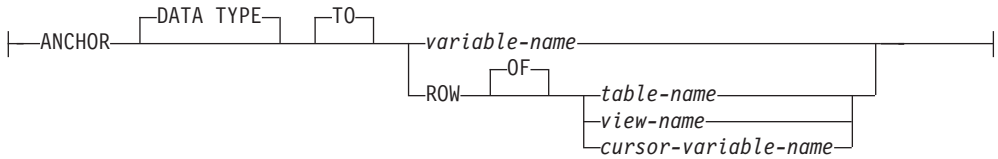
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the schema name of the cursor type does not refer to an existing schema
- CREATEIN privilege on the schema, if the schema name of the cursor type refers to an existing schema
- DBADM authority

Syntax



anchored-row-data-type:



Description

OR REPLACE

Specifies to replace the definition for the data type if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that functions and methods are invalidated instead of dropped when they have parameters or a return value defined with the data type being replaced. The existing definition must not be a structured type (SQLSTATE 42809). This option is ignored if a definition for the data type does not exist at the current server.

type-name

Names the type. The name, including the implicit or explicit qualifier, must not identify any other type (built-in or user-defined) that already exists at the current server. The unqualified name must not be the same as the name of a built-in data type or BOOLEAN, BINARY or VARBINARY (SQLSTATE 42918).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a type-name (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN,

UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators. If a two-part type-name is specified, the schema name must not begin with the characters 'SYS' (SQLSTATE 42939).

anchored-row-data-type
Identifies row information from another object used to determine the row type associated with the cursor type. The data type of the anchor object has the same limitations that apply to creating a row type.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies a global variable. The data type of the referenced variable must be a row type and is used as the row type associated with the cursor type.

ROW OF *table-name* **or** *view-name*
Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data types of the anchor object columns have the same limitations that apply to field data types. The row type associated with the cursor type is an unnamed row type.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following objects (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a *select-statement* where all the result columns are named.

If the cursor type of the cursor variable is not strongly-typed using a named row type, the row type associated with the cursor type is an unnamed row type.

row-type-name
Specifies the row type that will be used to check the row type of the result table of the *select-statement* assigned to a variable of the cursor type. The assignment fails if the type check fails (SQLSTATE 42821). If *row-type-name* is specified without a schema name, the row type is resolved by searching the schemas in the SQL path.

Rules

- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

Notes

- ***Cursor type usage:*** A cursor type can only be used as the data type of:
  - A local variable in a compound SQL (compiled) statement
  - A parameter of an SQL routine
  - The returns type of an SQL function

  - A global variable
- A variable or parameter defined with a cursor type can only be used in compound SQL (compiled) statements
- A variable or parameter that has a strongly-typed cursor type must not be used to assign cursor values that are based on a *statement-name* instead of a *select-statement*
- A user-defined cursor type with an associated row type is a strongly-typed cursor type; otherwise, it is a weakly-typed cursor type.

Examples

- *Example 1:* Create a cursor type that can be used with any cursor.

```
CREATE TYPE EMPCURSOR AS CURSOR
```

- *Example 2:* Create a strongly-typed cursor type that is based on the row data type DEPTROW:

```
CREATE TYPE DEPTCURSOR AS DEPTROW CURSOR
```

CREATE TYPE (distinct)

The CREATE TYPE (Distinct) statement defines a distinct type. The distinct type is always sourced on one of the built-in data types and can be defined to use strong type or weak type rules..

Successful execution of the statement that defines a strongly typed distinct type also generates functions to cast between the distinct type and its source type and, optionally, generates support for the comparison operators (=, <>, <, <=, >, and >=) for use with the distinct type. Successful execution of the statement that defines a weakly typed distinct type does not generate any functions.

Invocation

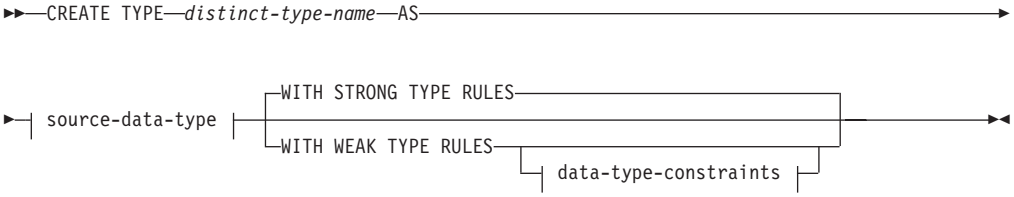
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include as least one of the following authorities:

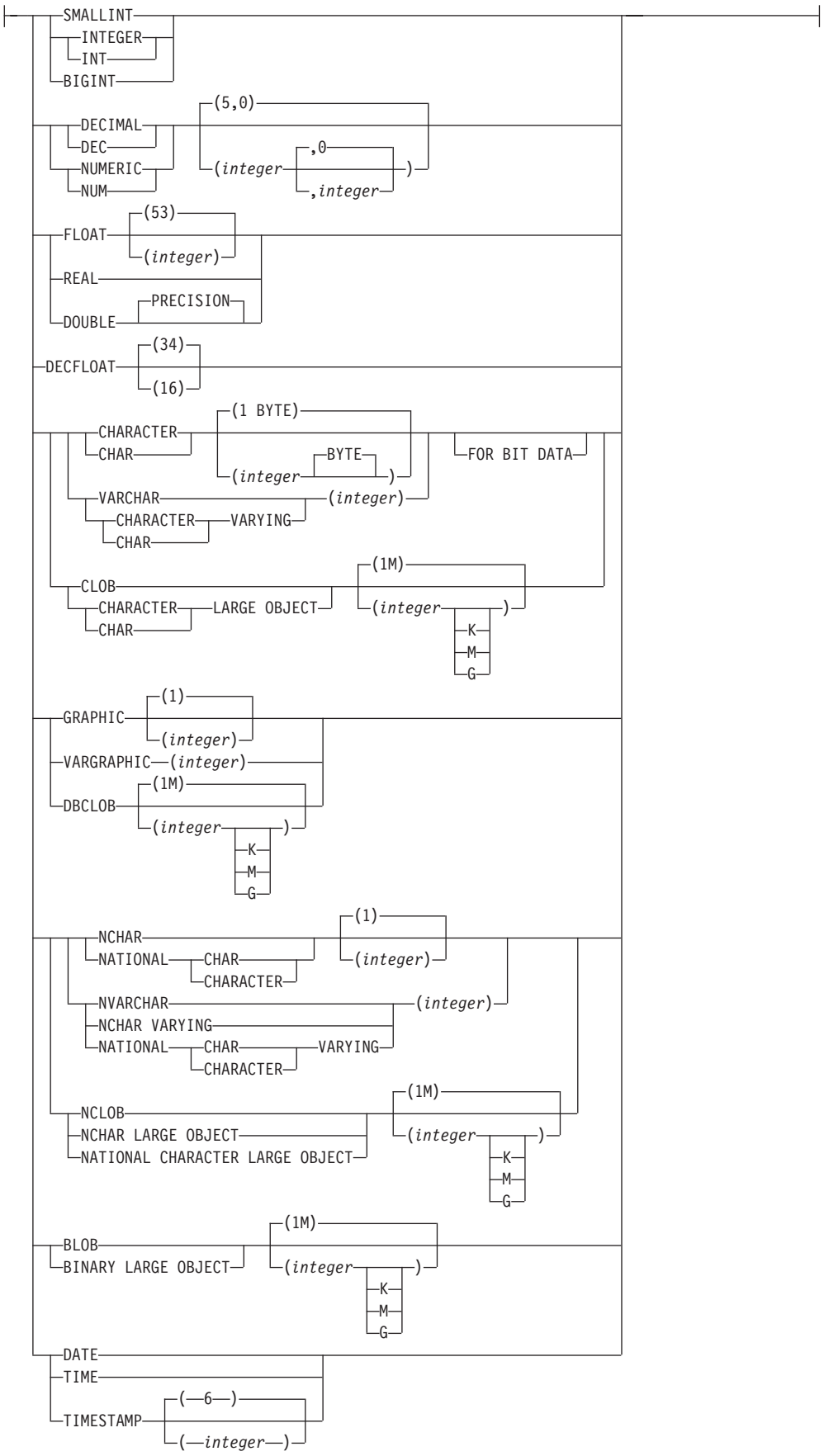
- IMPLICIT_SCHEMA authority on the database, if the schema name of the distinct type does not refer to an existing schema
- CREATEIN privilege on the schema, if the schema name of the distinct type refers to an existing schema
- DBADM authority

Syntax

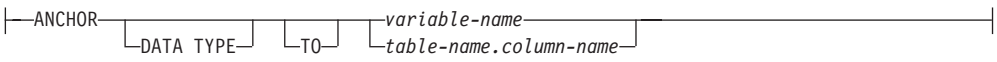


source-data-type:

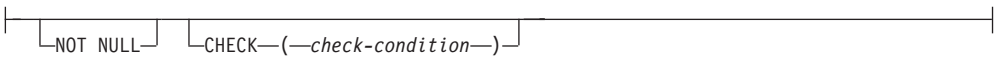
built-in-type:


SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer
,0
,integer
)
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
DECFLOAT
(34)
(16)
CHARACTER
CHAR
(1 BYTE)
(integer
BYTE
)
FOR BIT DATA
VARCHAR
CHARACTER
VARYING
CHAR
(integer)
CLOB
CHARACTER
LARGE OBJECT
CHAR
(1M)
(integer
K
M
G
)
GRAPHIC
(1)
(integer)
VARGRAPHIC
(integer)
DBCLOB
(1M)
(integer
K
M
G
)
NCHAR
NATIONAL
CHAR
CHARACTER
(1)
(integer)
NVARCHAR
NCHAR VARYING
NATIONAL
CHAR
CHARACTER
VARYING
(integer)
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
(1M)
(integer
K
M
G
)
BLOB
BINARY LARGE OBJECT
(1M)
(integer
K
M
G
)
DATE
TIME
TIMESTAMP
(6)
(integer)

anchored-data-type:



data-type-constraints:



Description

distinct-type-name
Names the distinct type. The name, including the implicit or explicit qualifier, must not identify any other type (built-in or user-defined) that already exists at the current server. The unqualified name must not be the same as the name of a built-in data type or BOOLEAN, BINARY, or VARBINARY (SQLSTATE 42918). The unqualified name should also not be ARRAY, INTERVAL, or ROWID.

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *distinct-type-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

If a two-part *distinct-type-name* is specified, the schema name must not begin with the characters 'SYS' (SQLSTATE 42939).

source-data-type
Specifies the data type used as the basis for the internal representation of the distinct type. The data type must be a built-in data type. For more information about built-in data types, see "CREATE TABLE". The source data type cannot be of type XML or an ARRAY type (SQLSTATE 42601). For portability of applications across platforms, use the following recommended data type names:

- DOUBLE or REAL instead of FLOAT
- DECIMAL instead of NUMERIC
- VARCHAR, BLOB, or CLOB instead of LONG VARCHAR
- VARGRAPHIC or DBCLOB instead of LONG VARGRAPHIC

anchored-data-type
Identifies another object used to determine the data type. The data type of the anchor object is bound by the same limitations that apply when specifying the data type directly.

ANCHOR DATA TYPE TO
Indicates that an anchored data type is used to specify the data type.

variable-name
Identifies a global variable with a data type that is a built-in type other than ROW or CURSOR. The data type of the global variable is used as the source data type for the distinct type.

table-name.column-name
Identifies a column name of an existing table or view with a data type that must be specified as a built-in-type. The data type of the column is used as the source data type for the distinct type.

WITH STRONG TYPE RULES
Specifies that strong typing rules are used for operations where this data type is an operand including assignments and comparisons. This is the default.

WITH WEAK TYPE RULES
Specifies that weak typing rules are used for operations where this data type is an operand including assignments, comparisons, and function resolution. When values of a of a weakly typed distinct type are used, the data type is effectively treated as the specified *source-data-type* when processing the operation.

data-type-constraints
Defines constraints on the distinct type that are applied when values are assigned or cast to the distinct type.

NOT NULL
Prevents a value with this distinct type from having a null value. If NOT NULL is not specified, a value with this distinct type can have the null value.

CHECK (*check-condition***)**
Defines a data type check constraint. At any time, the check-condition must be true or unknown for every value with this data type. The *check-condition*is a form of the *search-condition* that conforms to the rules of table check constraints (SQLSTATE 426211) with the addition that the VALUE keyword is used to reference a value that is assigned or cast to the distinct type in the same way that a column name is referenced in a table check constraint. Note that the *check-condition* cannot reference global variables.

Rules

- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

Notes

- ***Privileges***: The definer of the user-defined type always receives the EXECUTE privilege WITH GRANT OPTION on all functions automatically generated for the distinct type.

  EXECUTE privilege on all functions automatically generated during the CREATE TYPE (Distinct) statement is granted to PUBLIC.
- Creating a distinct type with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- ***Additional generated functions***: When a strongly typed distinct type is created, the following functions are generated to cast to and from the source type:
  - One function to convert from the distinct type to the source type
  - One function to convert from the source type to the distinct type

– One function to convert from INTEGER to the distinct type if the source type is SMALLINT
– One function to convert from VARCHAR to the distinct type if the source type is CHAR
– One function to convert from VARGRAPHIC to the distinct type if the source type is GRAPHIC.

In general these functions will have the following format:

```
CREATE FUNCTION source-type-name (distinct-type-name)
  RETURNS source-type-name ...

CREATE FUNCTION distinct-type-name (source-type-name)
  RETURNS distinct-type-name ...
```

In cases in which the source type is a parameterized type, the function to convert from the distinct type to the source type will have as function name the name of the source type without the parameters (see Table 28 for details). The type of the return value of this function will include the parameters given on the CREATE TYPE (Distinct) statement. The function to convert from the source type to the distinct type will have an input parameter whose type is the source type including its parameters. For example,

```
CREATE TYPE T_SHOESIZE AS CHAR(2)
  WITH COMPARISONS

CREATE TYPE T_MILES AS DOUBLE
  WITH COMPARISONS
```

will generate the following functions:

```
FUNCTION CHAR (T_SHOESIZE) RETURNS CHAR (2)

FUNCTION T_SHOESIZE (CHAR (2))
RETURNS T_SHOESIZE

FUNCTION DOUBLE (T_MILES) RETURNS DOUBLE

FUNCTION T_MILES (DOUBLE) RETURNS T_MILES
```

The schema of the generated cast functions is the same as the schema of the distinct type. No other function with this name and with the same signature may already exist in the database (SQLSTATE 42710).

The following table gives the names of the functions to convert from the distinct type to the source type and from the source type to the distinct type for all predefined data types.

Table 28. CAST functions on distinct types

Source Type Name	Function Name	Parameter	Return-type
SMALLINT	*distinct-type-name*	SMALLINT	*distinct-type-name*
SMALLINT	*distinct-type-name*	INTEGER	*distinct-type-name*
SMALLINT	SMALLINT	*distinct-type-name*	SMALLINT
INTEGER	*distinct-type-name*	INTEGER	*distinct-type-name*
INTEGER	INTEGER	*distinct-type-name*	INTEGER
BIGINT	*distinct-type-name*	BIGINT	*distinct-type-name*
BIGINT	BIGINT	*distinct-type-name*	BIGINT
DECIMAL	*distinct-type-name*	DECIMAL (*p,s*)	*distinct-type-name*
DECIMAL	DECIMAL	*distinct-type-name*	DECIMAL (*p,s*)
NUMERIC	*distinct-type-name*	DECIMAL (*p,s*)	*distinct-type-name*

Table 28. CAST functions on distinct types (continued)

Source Type Name	Function Name	Parameter	Return-type
NUMERIC	DECIMAL	*distinct-type-name*	DECIMAL (*p*,*s*)
REAL	*distinct-type-name*	REAL	*distinct-type-name*
REAL	*distinct-type-name*	DOUBLE	*distinct-type-name*
REAL	REAL	*distinct-type-name*	REAL
FLOAT(*n*) where *n*<=24	*distinct-type-name*	REAL	*distinct-type-name*
FLOAT(*n*) where *n*<=24	*distinct-type-name*	DOUBLE	*distinct-type-name*
FLOAT(*n*) where *n*<=24	REAL	*distinct-type-name*	REAL
FLOAT(*n*) where *n*>24	*distinct-type-name*	DOUBLE	*distinct-type-name*
FLOAT(*n*) where *n*>24	DOUBLE	*distinct-type-name*	DOUBLE
FLOAT	*distinct-type-name*	DOUBLE	*distinct-type-name*
FLOAT	DOUBLE	*distinct-type-name*	DOUBLE
DOUBLE	*distinct-type-name*	DOUBLE	*distinct-type-name*
DOUBLE	DOUBLE	*distinct-type-name*	DOUBLE
DOUBLE PRECISION	*distinct-type-name*	DOUBLE	*distinct-type-name*
DOUBLE PRECISION	DOUBLE	*distinct-type-name*	DOUBLE
DECFLOAT	*distinct-type-name*	DECFLOAT(*n*)	*distinct-type-name*
DECFLOAT	DECFLOAT	*distinct-type-name*	DECFLOAT(*n*)
CHAR	*distinct-type-name*	CHAR (*n*)	*distinct-type-name*
CHAR	CHAR	*distinct-type-name*	CHAR (*n*)
CHAR	*distinct-type-name*	VARCHAR (*n*)	*distinct-type-name*
VARCHAR	*distinct-type-name*	VARCHAR (*n*)	*distinct-type-name*
VARCHAR	VARCHAR	*distinct-type-name*	VARCHAR (*n*)
CLOB	*distinct-type-name*	CLOB (*n*)	*distinct-type-name*
CLOB	CLOB	*distinct-type-name*	CLOB (*n*)
GRAPHIC	*distinct-type-name*	GRAPHIC (*n*)	*distinct-type-name*
GRAPHIC	GRAPHIC	*distinct-type-name*	GRAPHIC (*n*)
GRAPHIC	*distinct-type-name*	VARGRAPHIC (*n*)	*distinct-type-name*
VARGRAPHIC	*distinct-type-name*	VARGRAPHIC (*n*)	*distinct-type-name*
VARGRAPHIC	VARGRAPHIC	*distinct-type-name*	VARGRAPHIC (*n*)
DBCLOB	*distinct-type-name*	DBCLOB (*n*)	*distinct-type-name*
DBCLOB	DBCLOB	*distinct-type-name*	DBCLOB (*n*)
BLOB	*distinct-type-name*	BLOB (*n*)	*distinct-type-name*
BLOB	BLOB	*distinct-type-name*	BLOB (*n*)
DATE	*distinct-type-name*	DATE	*distinct-type-name*
DATE	DATE	*distinct-type-name*	DATE
TIME	*distinct-type-name*	TIME	*distinct-type-name*
TIME	TIME	*distinct-type-name*	TIME
TIMESTAMP	*distinct-type-name*	TIMESTAMP(*p*	*distinct-type-name*
TIMESTAMP	TIMESTAMP	*distinct-type-name*	TIMESTAMP(*p*)

Table 28. CAST functions on distinct types (continued)

Source Type Name	Function Name	Parameter	Return-type

Note: NUMERIC and FLOAT are not recommended when creating a user-defined type for a portable application. DECIMAL and DOUBLE should be used instead.

The functions described in the preceding table and the comparison operator functions are the only functions that are generated automatically when distinct types are defined. Consequently, none of the built-in functions (AVG, MAX, LENGTH, and so on) are supported for strongly typed distinct types until the CREATE FUNCTION statement is used to register user-defined functions for the strongly typed distinct type, and those user-defined functions are sourced on the appropriate built-in functions. In particular, note that it is possible to register user-defined functions that are sourced on the built-in column functions.

When a strongly typed distinct type is created, system-generated comparison operators are created when the source type supports comparisons . Creation of these comparison operators will generate entries in the SYSCAT.ROUTINES catalog view for the new functions.

The schema name of the distinct type must be included in the SQL path or the FUNCPATH BIND option for successful use of these operators and cast functions in SQL statements.

- When a weakly typed distinct type is created, no additional functions need to be generated or created because the weak type rules allow a weakly typed distinct type to be used in the same context where the source type can be used.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - CREATE DISTINCT TYPE can be specified in place of CREATE TYPE
  - The LONG VARCHAR and LONG VARGRAPHIC data types and cast functions are supported but are deprecated and might be removed in a future release. The WITH COMPARISONS clause continues to not support the LONG VARCHAR and LONG VARGRAPHIC data types.
  - The WITH COMPARISONS clause, which specifies that system-generated comparison operators are to be created for comparing two instances of the distinct type, can be specified as the last clause of the statement if WITH WEAK TYPE RULES is not specified. Use WITH COMPARISONS only if it is required for compatibility with earlier versions of products in the DB2 family. If the source data type is either BLOB, CLOB, or DBCLOB and WITH COMPARISONS is specified, a warning occurs as in previous releases.
  - ALLOW NULL, or just NULL, can be specified as the opposite of NOT NULL. This is the default nullability characteristic of the distinct type if neither the ALLOW NULL clause nor the NOT NULL clause are specified. Specification of ALLOW NULL is not considered to define a data type constraint for the distinct type.

Examples

- *Example 1:* Create a strongly typed distinct type named SHOESIZE that is based on an INTEGER data type.

```
CREATE TYPE SHOESIZE AS INTEGER
```

This will also result in the creation of comparison operators (=, <>, <, <=, >, >=) and cast functions INTEGER(SHOESIZE) returning INTEGER and SHOESIZE(INTEGER) returning SHOESIZE.

- *Example 2:* Create a strongly typed distinct type named MILES that is based on a DOUBLE data type.

  ```
  CREATE TYPE MILES AS DOUBLE
  ```

  This will also result in the creation of comparison operators (=, <>, <, =, >, >=) and cast functions DOUBLE(MILES) returning DOUBLE and MILES(DOUBLE) returning MILES.

- *Example 3:* Create a weakly typed distinct type named BONUS that is based on an INTEGER data type and represents a percentage which cannot exceed 100.

  ```
  CREATE TYPE BONUS AS INTEGER WITH WEAK TYPE RULES
        CHECK(VALUE >= 0 AND VALUE <= 100)
  ```

  Because it is defined with weak type rules, comparison and cast functions are not generated for the weakly typed distinct type called BONUS.

- *Example 4:* Create a weakly typed distinct type named SALARY that is based on a DOUBLE data type which cannot be NULL and where the upper range is limited to less than one hundred thousand.

  ```
  CREATE TYPE SALARY AS DOUBLE WITH WEAK TYPE RULES
        NOT NULL CHECK(VALUE < 100000)
  ```

CREATE TYPE (row)

The CREATE TYPE (row) statement defines a row type. A row type includes one or more fields with associated data types that make up a row of data.

Invocation

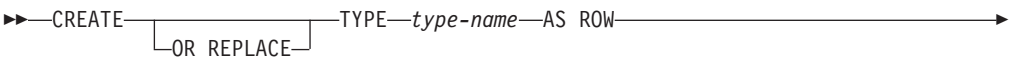
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

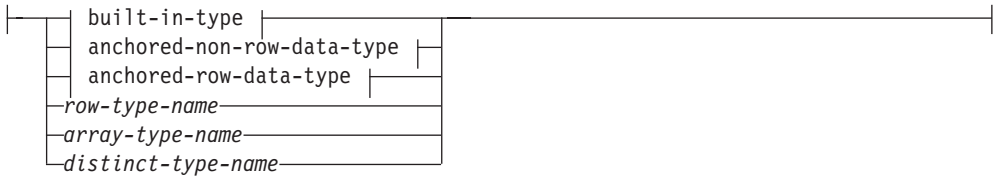
- IMPLICIT_SCHEMA authority on the database, if the schema name of the row type does not refer to an existing schema
- CREATEIN privilege on the schema, if the schema name of the row type refers to an existing schema
- DBADM authority

Syntax

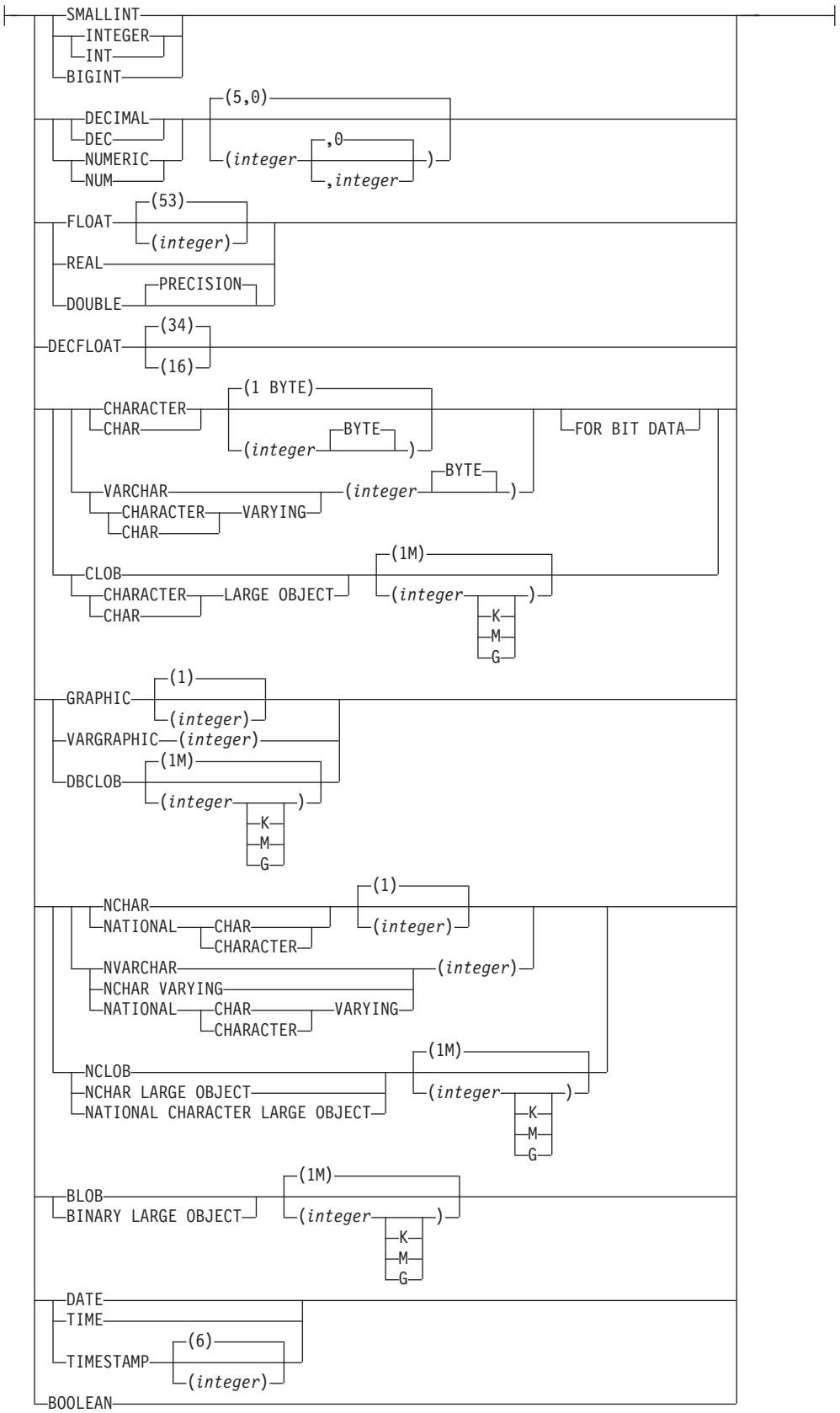


field-definition:

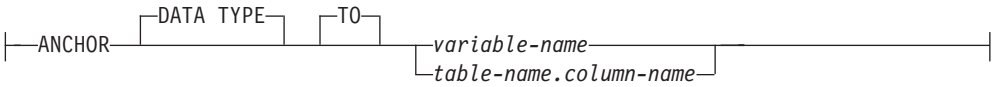
data-type:



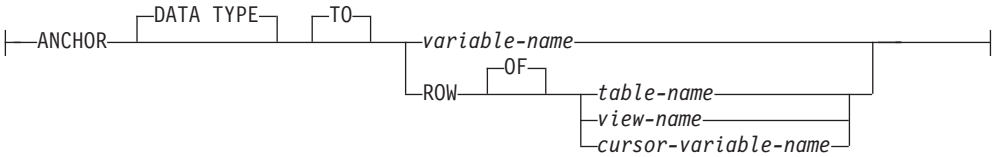
built-in-type:


SMALLINT
INTEGER
INT
BIGINT
DECIMAL
DEC
NUMERIC
NUM
(5,0)
(integer)
,0
,integer
FLOAT
(53)
(integer)
REAL
DOUBLE
PRECISION
DECFLOAT
(34)
(16)
CHARACTER
CHAR
(1 BYTE)
(integer)
BYTE
FOR BIT DATA
VARCHAR
CHARACTER
VARYING
CHAR
CLOB
CHARACTER
LARGE OBJECT
CHAR
(1M)
K
M
G
GRAPHIC
(1)
(integer)
VARGRAPHIC
DBCLOB
NCHAR
NATIONAL
CHAR
CHARACTER
NVARCHAR
NCHAR VARYING
NCLOB
NCHAR LARGE OBJECT
NATIONAL CHARACTER LARGE OBJECT
BLOB
BINARY LARGE OBJECT
DATE
TIME
TIMESTAMP
(6)
BOOLEAN

anchored-non-row-data-type:



anchored-row-data-type:



Description

OR REPLACE
Specifies to replace the definition for the data type if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that functions and methods are invalidated instead of dropped when they have parameters or a return value defined with the data type being replaced. The existing definition must not be a structured type (SQLSTATE 42809). This option is ignored if a definition for the data type does not exist at the current server.

type-name
Names the type. The name, including the implicit or explicit qualifier, must not identify any other type (built-in, structured, array, row, or distinct) already described in the catalog. The unqualified name must not be the same as the name of a built-in data type or BOOLEAN (SQLSTATE 42918).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *type-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

If a two-part *type-name* is specified, the schema name cannot begin with 'SYS'; otherwise, an error is returned (SQLSTATE 42939).

field-definition
Defines the fields of the row type.

field-name
Specifies the name of a field within the row type. The name cannot be the same as any other field of this row type (SQLSTATE 42711).

data-type
Specifies the data type of the field.

built-in-type
Specifies a built-in data type. See "CREATE TABLE" for the description of built-in data types. Built-in types include the data types described in “CREATE TABLE”, other than reference, SYSPROC.DB2SECURITYLABEL, XML, or user-defined types (SQLSTATE 429C2).

row-type-name
Specifies the name of a user-defined row type. If a *row-type-name* is specified without a schema name, the *row-type-name* is resolved by

searching the schemas in the SQL path. Row types can be nested as field types of a row type with a maximum nesting level of sixteen.

array-type-name
Specifies an array type. If an *array-type-name* is specified without a schema name, the *array-type-name* is resolved by searching the schemas in the SQL path. Array types can be nested as field types of a row type with a maximum nesting level of sixteen.

distinct-type-name
Specifies a user-defined distinct data type. The specified distinct type cannot have any data type constraints (SQLSTATE 429C5).

anchored-non-row-data-type
Identifies another object used to determine the data type. The data type of the anchor object is has the same limitations that apply when specifying the data type directly.

ANCHOR DATA TYPE TO
Indicates that an anchored data type is used to specify the data type.

variable-name
Identifies a global variable with a data type that is a supported row field data type. The data type of the global variable is used as the data type for the field.

table-name.column-name
Identifies a column name of an existing table or view with a data type that is a built-in-type or a distinct type. The data type of the column is used as the data type for the field.

anchored-row-data-type
Identifies row information from another object to use as the fields of the row.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies a global variable. The data type of the referenced variable must be a row type.

ROW OF *table-name* **or** *view-name*
Specifies a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data types of the anchor object columns have the same limitations that apply to field data types.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following objects (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a *select-statement* where all the result columns are named.

Rules

- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row

definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

Notes

- ***Row type usage:*** A row type can only be used as the data type of:
  - A local variable in a compound SQL (compiled) statement
  - A parameter of an SQL routine
  - The returns type of an SQL function
  - The element of an array type
  - A user-defined cursor type
  - A global variable
- A variable or parameter defined with a row type can only be used in compound SQL (compiled) statements

Example

- Create a row type based on the columns of the DEPARTMENT table.

```
CREATE TYPE DEPTROW AS ROW (DEPTNO   VARCHAR(3),
                            DEPTNAME VARCHAR(29),
                            MGRNO    CHAR(6),
                            ADMRDEPT CHAR(3),
                            LOCATION CHAR(16))
```

CREATE TYPE (structured)

The CREATE TYPE statement defines a user-defined structured type.

A user-defined structured type can include zero or more attributes. A structured type can be a subtype allowing attributes to be inherited from a supertype. Successful execution of the statement generates methods, for retrieving and updating values of attributes. Successful execution of the statement also generates functions, for constructing instances of a structured type used in a column, for casting between the reference type and its representation type, and for supporting the comparison operators (=, <>, <, <=, >, and >=) on the reference type.

The CREATE TYPE statement also defines any method specifications for user-defined methods to be used with the user-defined structured type.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

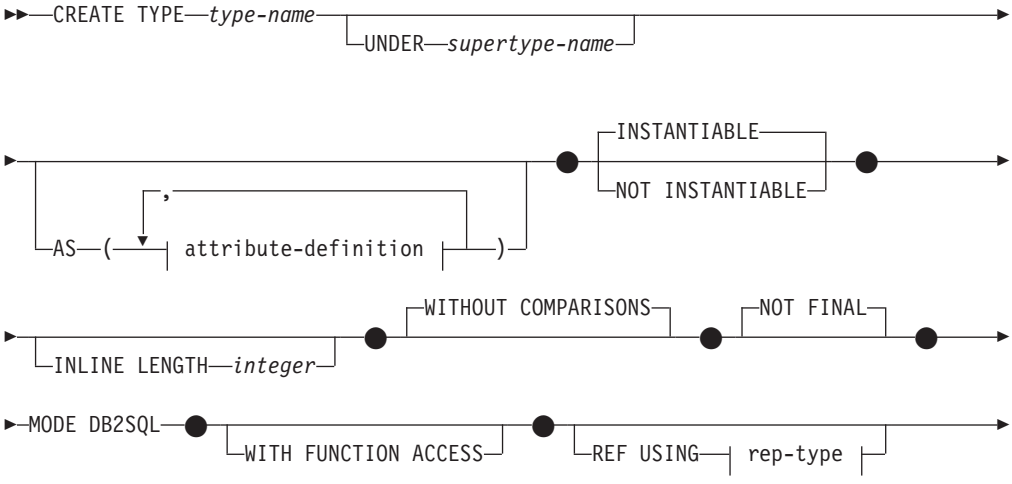
Authorization

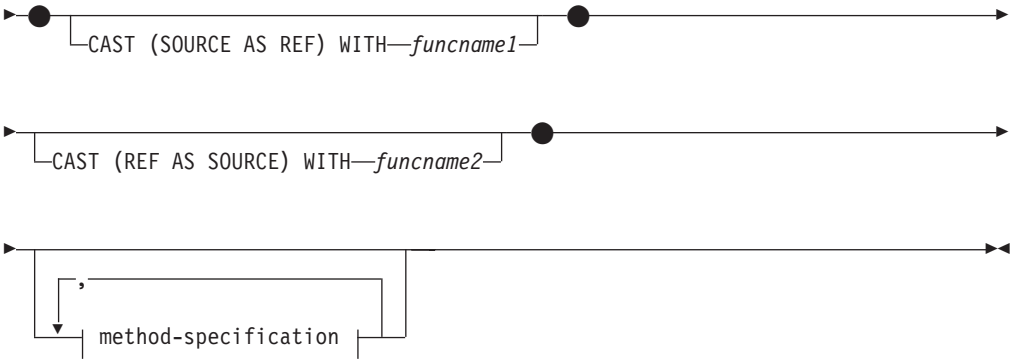
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the schema name of the type does not refer to an existing schema
- CREATEIN privilege on the schema, if the schema name of the type refers to an existing schema
- DBADM authority

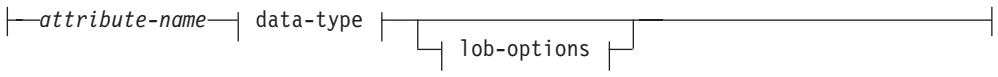
If UNDER is specified, and the authorization ID of the statement is not the same as the owner of the root type of the type hierarchy, DBADM authority is required.

Syntax

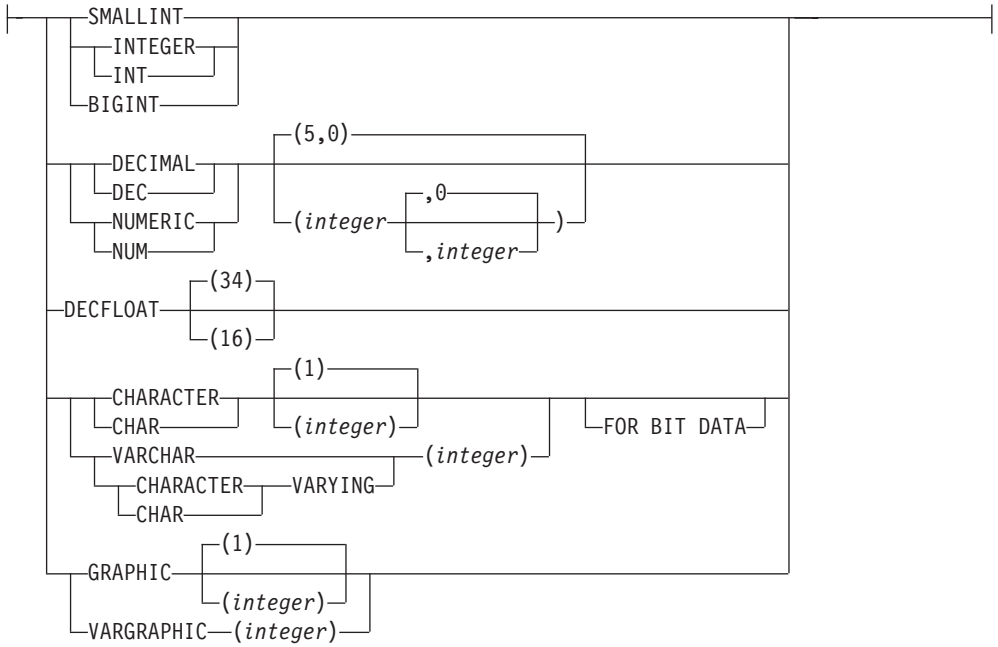




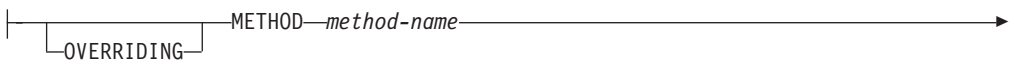
attribute-definition:

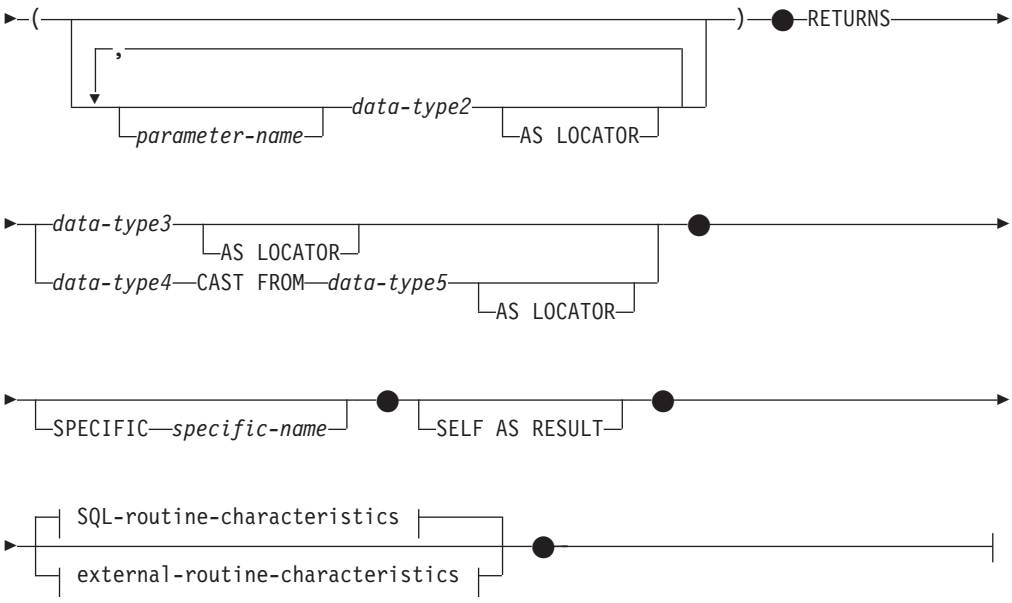


rep-type:

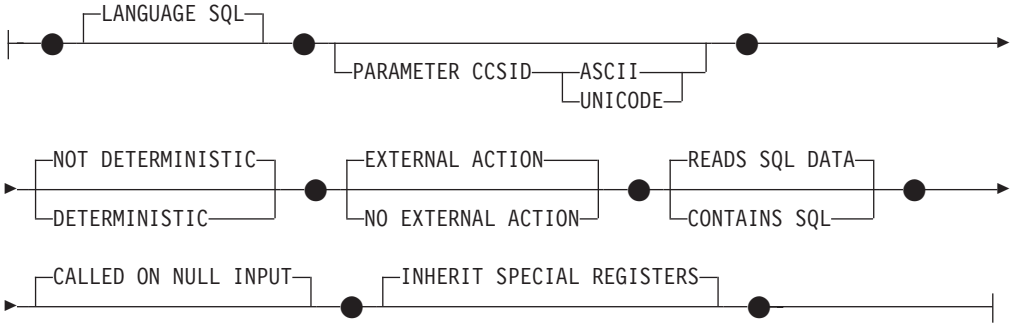


method-specification:

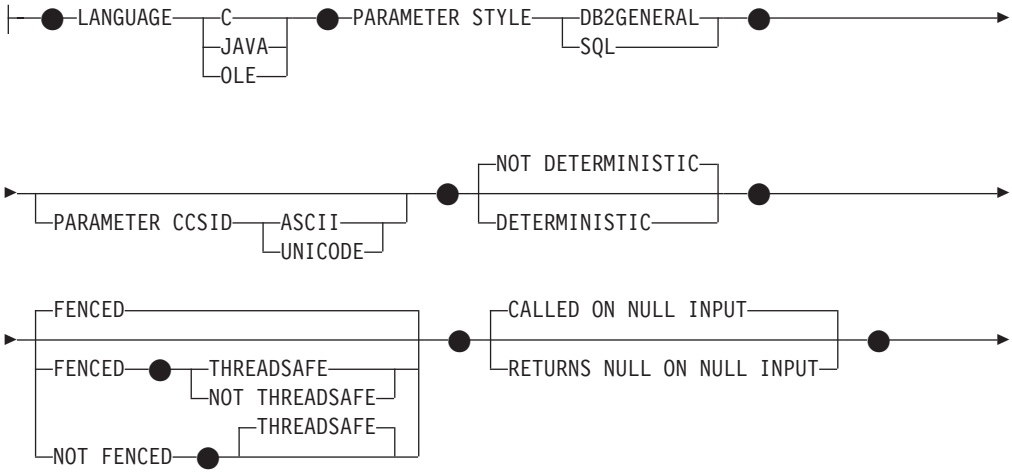


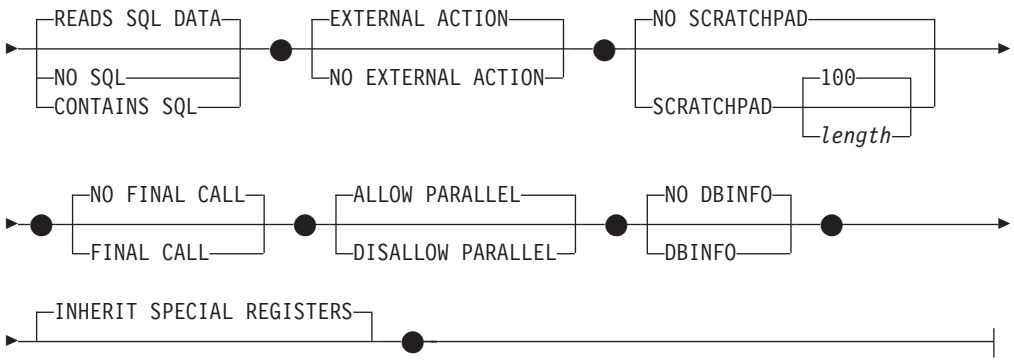


SQL-routine-characteristics:



external-routine-characteristics:





Description

type-name
Names the type. The name, including the implicit or explicit qualifier, must not identify any other type (built-in, structured, or distinct) that already exists at the current server. The unqualified name must not be the same as the name of a built-in data type, BINARY, VARBINARY, or BOOLEAN (SQLSTATE 42918). The unqualified name should also not be ARRAY, INTERVAL, or ROWID. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *type-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

If a two-part *type-name* is specified, the schema name must not begin with the characters 'SYS' (SQLSTATE 42939).

UNDER *supertype-name*
Specifies that this structured type is a subtype under the specified *supertype-name*. The *supertype-name* must identify an existing structured type (SQLSTATE 42704). If *supertype-name* is specified without a schema name, the type is resolved by searching the schemas on the SQL path. The structured type includes all the attributes of the supertype followed by the additional attributes given in the *attribute-definition*.

attribute-definition
Defines the attributes of the structured type.

attribute-name
The name of an attribute. The *attribute-name* cannot be the same as any other attribute of this structured type or any supertype of this structured type (SQLSTATE 42711).

A number of names used as keywords in predicates are reserved for system use, and cannot be used as an *attribute-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

data-type
The data type of the attribute. It is one of the data types listed under “CREATE TABLE”, other than XML or a weakly typed distinct type (SQLSTATE 42601). The data type must identify an existing data type

(SQLSTATE 42704). If *data-type* is specified without a schema name, the type is resolved by searching the schemas on the SQL path. The description of various data types is given in "CREATE TABLE". If the attribute data type is a reference type, the target type of the reference must be a structured type that exists, or is created by this statement (SQLSTATE 42704).

To prevent type definitions that would, at run time, permit an instance of the type to directly or indirectly contain another instance of the same type or one of its subtypes, a type cannot be defined such that one of its attribute types directly or indirectly uses itself (SQLSTATE 428EP).

lob-options
Specifies the options associated with LOB types (or distinct types based on LOB types). For a detailed description of *lob-options*, see "CREATE TABLE".

INSTANTIABLE or NOT INSTANTIABLE
Determines whether an instance of the structured type can be created. Implications of not instantiable structured types are:

- no constructor function is generated for a non-instantiable type
- a non-instantiable type cannot be used as the type of a table or view (SQLSTATE 428DP)
- a non-instantiable type can be used as the type of a column (only null values or instances of instantiable subtypes can be inserted into the column.

To create instances of a non-instantiable type, instantiable subtypes must be created. If NOT INSTANTIABLE is specified, no instance of the new type can be created.

INLINE LENGTH *integer*
This option indicates the maximum size (in bytes) of a structured type column instance to store inline with the rest of the values in the row of a table. Instances of a structured type or its subtypes, that are larger than the specified inline length, are stored separately from the base table row, similar to the way that LOB values are handled.

If the specified INLINE LENGTH is smaller than the size of the result of the constructor function for the newly-created type (32 bytes plus 10 bytes per attribute) and smaller than 292 bytes, an error results (SQLSTATE 429B2). Note that the number of attributes includes all attributes inherited from the supertype of the type.

The INLINE LENGTH for the type, whether specified or a default value, is the default inline length for columns that use the structured type. This default can be overridden at CREATE TABLE time.

INLINE LENGTH has no meaning when the structured type is used as the type of a typed table.

The default INLINE LENGTH for a structured type is calculated by the system. In the formulae that follow, the following terms are used:

short attribute
refers to an attribute with any of the following data types: SMALLINT, INTEGER, BIGINT, REAL, DOUBLE, FLOAT, DATE, or TIME. Also included are distinct types or reference types based on these types.

non-short attribute
refers to an attribute of any of the remaining data types, or distinct types based on those data types.

The system calculates the default inline length as follows:

1. Determine the added space requirements for non-short attributes using the following formula:

   *space_for_non_short_attribute*s = SUM(*attributelength* + n)

   n is defined as:

   - 0 bytes for nested structured type attributes
   - 2 bytes for non-LOB attributes
   - 9 bytes for LOB attributes

   attributelength is based on the data type specified for the attribute as shown in Table 29.

2. Calculate the total default inline length using the following formula:

   default_length(*structured_type*) = (*number_of_attributes* * 10) + 32 + *space_for_non-short_attributes*

   number_of_attributes is the total number of attributes for the structured type, including attributes that are inherited from its supertype. However, *number_of_attributes* does not include any attributes defined for any subtype of *structured_type*.

Table 29. Byte Counts for Attribute Data Types

Attribute Data Type	Byte Count
DECIMAL	The integral part of (*p* / 2) + 1, where *p* is the precision
DECFLOAT(*n*)	If *n* is 16, the byte count is 8; if *n* is 34, the byte count is 16
CHAR(*n*)	*n*
VARCHAR(*n*)	*n*
GRAPHIC(*n*)	*n* * 2
VARGRAPHIC(*n*)	*n* * 2
TIMESTAMP	10
LOB type	Each LOB attribute has a LOB descriptor in the structured type instance that points to the location of the actual value. The size of the descriptor varies according to the maximum length defined for the LOB attribute (see Table 30.
Distinct type	Length of the source type of the distinct type
Reference type	Length of the built-in data type on which the reference type is based
Structured type	inline_length(*attribute_type*)

Table 30. LOB Descriptor Size as a Function of the Maximum LOB Length

Maximum LOB Length	LOB Descriptor Size
1024	68
8192	92
65 536	116
524 000	140
4 190 000	164
134 000 000	196
536 000 000	220
1 070 000 000	252

Table 30. LOB Descriptor Size as a Function of the Maximum LOB Length (continued)

Maximum LOB Length	LOB Descriptor Size
1 470 000 000	276
2 147 483 647	312

WITHOUT COMPARISONS
Indicates that there are no comparison functions supported for instances of the structured type.

NOT FINAL
Indicates that the structured type may be used as a supertype.

MODE DB2SQL
This clause is required and allows for direct invocation of the constructor function on this type.

WITH FUNCTION ACCESS
Indicates that all methods of this type and its subtypes, including methods created in the future, can be accessed using functional notation. This clause can be specified only for the root type of a structured type hierarchy (the UNDER clause is not specified) (SQLSTATE 42613). This clause is provided to allow the use of functional notation for those applications that prefer this form of notation over method invocation notation.

REF USING *rep-type*
Defines the built-in data type used as the representation (underlying data type) for the reference type of this structured type and all its subtypes. This clause can only be specified for the root type of a structured type hierarchy (UNDER clause is not specified) (SQLSTATE 42613). The *rep-type* cannot be a REAL, FLOAT, DECFLOAT, BLOB, CLOB, DBCLOB, array type, or structured type, and must have a length less than or equal to 32 672 bytes (SQLSTATE 42613).

If this clause is not specified for the root type of a structured type hierarchy, then REF USING VARCHAR(16) FOR BIT DATA is assumed.

CAST (SOURCE AS REF) WITH *funcname1*
Defines the name of the system-generated function that casts a value with the data type *rep-type* to the reference type of this structured type. A schema name must not be specified as part of *funcname1* (SQLSTATE 42601). The cast function is created in the same schema as the structured type. If the clause is not specified, the default value for *funcname1* is *type-name* (the name of the structured type). A function signature matching *funcname1(rep-type)* must not already exist in the same schema (SQLSTATE 42710).

CAST (REF AS SOURCE) WITH *funcname2*
Defines the name of the system-generated function that casts a reference type value for this structured type to the data type *rep-type*. A schema name must not be specified as part of *funcname2* (SQLSTATE 42601). The cast function is created in the same schema as the structured type. If the clause is not specified, the default value for *funcname2* is *rep-type* (the name of the representation type).

method-specification
Defines the methods for this type. A method cannot actually be used until it is given a body with a CREATE METHOD statement (SQLSTATE 42884).

OVERRIDING
Specifies that the method being defined overrides a method of a supertype of the type being defined. Overriding enables one to re-implement

methods in subtypes, thereby providing more specific functionality. Overriding is not supported for the following types of methods:

- Table and row methods
- External methods declared with PARAMETER STYLE JAVA
- Methods that can be used as predicates in an index extension
- System-generated mutator or observer methods

Attempting to override such a method will result in an error (SQLSTATE 42745).

If a method is to be a valid overriding method, there must already exist one original method for one of the proper supertypes of the type being defined, and the following relationships must exist between the overriding method and the original method:

- The method name of the method being defined and the original method are equivalent.
- The method being defined and the original method have the same number of parameters.
- The data type of each parameter of the method being defined and the data type of the corresponding parameters of the original method are identical. This requirement excludes the implicit SELF parameter.

If such an original method does not exist, an error is returned (SQLSTATE 428FV).

The overriding method inherits the following attributes from the original method:

- Language
- Determinism indication
- External action indication
- An indication whether this method should be called if any of its arguments is the null value
- Result cast (if specified in the original method)
- SELF AS RESULT indication
- The SQL-data access or CONTAINS SQL indication
- For external methods:
  - Parameter style
  - Locator indication of the parameters and of the result (if specified in the original method)
  - FENCED, SCRATCHPAD, FINAL CALL, ALLOW PARALLEL, and DBINFO indication
  - INHERIT SPECIAL REGISTER and THREADSAFE indication

method-name

Names the method being defined. It must be an unqualified SQL identifier (SQLSTATE 42601). The method name is implicitly qualified with the schema used for CREATE TYPE.

A number of names used as keywords in predicates are reserved for system use, and cannot be used as a *method-name* (SQLSTATE 42939). The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH, and the comparison operators.

In general, the same name can be used for more than one method if there is some difference in their signatures.

parameter-name
Identifies the parameter name. It cannot be SELF, which is the name for the implicit subject parameter of a method (SQLSTATE 42734). If the method is an SQL method, all its parameters must have names (SQLSTATE 42629). If the method being declared overrides another method, the parameter name must be exactly the same as the name of the corresponding parameter of the overridden method; otherwise, an error is returned (SQLSTATE 428FV).

data-type2
Specifies the data type of each parameter. One entry in the list must be specified for each parameter that the method will expect to receive. No more than 90 parameters are allowed, including the implicit SELF parameter. If this limit is exceeded, an error is raised (SQLSTATE 54023).

You can specify SQL data types and abbreviations that can be specified as a column type in the CREATE TABLE statement, and that have equivalents in the language that is being used to write the method. For details on the mapping between SQL data types and host language data types, see the topic that pertains to your language from the following list of related topics.

Note: If the SQL data type in question is a structured type, there is no default mapping to a host language data type. A user-defined transform function must be used to create a mapping between the structured type and the host language data type.

DECIMAL (or NUMERIC) and decimal floating-point are invalid with LANGUAGE C and OLE (SQLSTATE 42815).

XML data types cannot be used (SQLSTATE 42815).

REF may be specified, but it does not have a defined scope. Inside the body of the method, a reference-type can be used in a path-expression only by first casting it to have a scope. Similarly, a reference returned by a method can be used in a path-expression only by first casting it to have a scope.

AS LOCATOR
For LOB types or distinct types which are based on a LOB type, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be passed to the method instead of the actual value. This saves greatly in the number of bytes passed to the method, and may save as well in performance, particularly in the case where only a few bytes of the value are actually of interest to the method.

An error is raised (SQLSTATE 42601) if AS LOCATOR is specified for a type other than a LOB or a distinct type based on a LOB.

If the method is FENCED, or if LANGUAGE is SQL, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

If the method being declared overrides another method, the AS LOCATOR indication of the parameter must match exactly the AS LOCATOR indication of the corresponding parameter of the overridden method (SQLSTATE 428FV).

If the method being declared overrides another method, the FOR BIT DATA indication of each parameter must match exactly the FOR BIT DATA indication of the corresponding parameter of the overridden method. (SQLSTATE 428FV).

RETURNS
This mandatory clause identifies the method's result.

data-type3
Specifies the data type of the method's result. In this case, exactly the same considerations apply as for the parameters of methods specified in the description for data-type2.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be passed from the method instead of the actual value.

An error is raised (SQLSTATE 42601) if AS LOCATOR is specified for a type other than a LOB or a distinct type based on a LOB.

If the method is FENCED, or if LANGUAGE is SQL, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

If the method being defined overrides another method, this clause cannot be specified (SQLSTATE 428FV).

If the method overrides another method, *data-type3* must be a subtype of the data type of the result of the overridden method if this data type is a structured type; otherwise both data types must be identical (SQLSTATE 428FV).

data-type4 **CAST FROM** *data-type5*
Specifies the data type of the method's result.

This clause is used to return a different data type to the invoking statement from the data type returned by the method code. The *data-type5* must be castable to the *data-type4* parameter. If it is not castable, an error is returned (SQLSTATE 42880).

Because the length, precision, or scale for *data-type4* can be inferred from *data-type5*, it is not necessary (but still permitted) to specify the length, precision, or scale for parameterized types specified for *data-type4*. Instead, empty parentheses can be used, such as VARCHAR(), for example. FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

A distinct type is not valid as the type specified in *data-type5* (SQLSTATE 42815). XML is not valid as the type specified in *data-type4* or *data-type5* (SQLSTATE 42815).

The cast operation is also subject to runtime checks that might result in conversion errors being returned.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be passed from the method instead of the actual value.

An error is raised (SQLSTATE 42601) if AS LOCATOR is specified for a type other than a LOB or a distinct type based on a LOB.

If the method is FENCED, or if LANGUAGE is SQL, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

If the method being defined overrides another method, this clause cannot be specified (SQLSTATE 428FV).

If the method being defined overrides another method, the FOR BIT DATA clause cannot be specified (SQLSTATE 428FV).

SPECIFIC *specific-name*
Provides a unique name for the instance of the method that is being defined. This specific name can be used when creating the method body or dropping the method. It can never be used to invoke the method. The unqualified form of *specific-name* is an SQL identifier (with a maximum length of 18). The qualified form is a schema-name followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another specific method name that exists at the application server; otherwise an error is raised (SQLSTATE 42710).

The *specific-name* may be the same as an existing *method-name*.

If no qualifier is specified, the qualifier that was used for *type-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of *type-name* or an error is raised (SQLSTATE 42882).

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmmssxxx.

SELF AS RESULT
Identifies this method as a type-preserving method, which is defined as follows:

- The declared return type must be the same as the declared subject-type (SQLSTATE 428EQ).
- When an SQL statement is compiled and resolves to a type preserving method, the static type of the result of the method is the same as the static type of the subject argument.
- The method must be implemented in such a way that the dynamic type of the result is the same as the dynamic type of the subject argument (SQLSTATE 2200G), and the result cannot be NULL (SQLSTATE 22004).

If the method being defined overrides another method, this clause cannot be specified (SQLSTATE 428FV).

SQL-routine-characteristics
Specifies the characteristics of the method body that will be defined for this type using CREATE METHOD.

LANGUAGE SQL
This clause is used to indicate that the method is written in SQL with a single RETURN statement. The method body is specified using the CREATE METHOD statement.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the SQL method. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE
Specifies that character data is in UTF-8, and that graphic data is in UCS-2. If the database is not a Unicode database, PARAMETER CCSID UNICODE cannot be specified (SQLSTATE 56031).

NOT DETERMINISTIC or DETERMINISTIC
This optional clause specifies whether the method always returns the same results for given argument values (DETERMINISTIC) or whether the method depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC method must always return the same result from successive invocations with identical inputs. Optimizations taking advantage of the fact that identical inputs always produce the same results are prevented by specifying NOT DETERMINISTIC. NOT DETERMINISTIC must be explicitly or implicitly specified if the body of the method accesses a special register, or calls another non-deterministic routine (SQLSTATE 428C2).

EXTERNAL ACTION or NO EXTERNAL ACTION
This optional clause specifies whether or not the method takes some action that changes the state of an object not managed by the database manager. Optimizations that assume methods have no external impacts are prevented by specifying EXTERNAL ACTION. For example: sending a message, ringing a bell, or writing a record to a file.

READS SQL DATA or CONTAINS SQL
Indicates what type of SQL statements can be executed. Because the SQL statement supported is the RETURN statement, the distinction has to do with whether or not the expression is a subquery.

READS SQL DATA
Indicates that SQL statements that do not modify SQL data can be executed by the method (SQLSTATE 42985). Nicknames cannot be referenced in the SQL statement (SQLSTATE 42997).

CONTAINS SQL
Indicates that SQL statements that neither read nor modify SQL data can be executed by the method (SQLSTATE 42985).

CALLED ON NULL INPUT
This optional clause indicates that regardless of whether any arguments are null, the user-defined method is called. It can return a null value or a normal (non-null) value. However, responsibility for testing for null argument values lies with the method.

If the method being defined overrides another method, this clause cannot be specified (SQLSTATE 428FV).

NULL CALL can be used as a synonym for CALLED ON NULL INPUT.

INHERIT SPECIAL REGISTERS
This optional clause specifies that updatable special registers in the method will inherit their initial values from the environment of the invoking statement. For a method invoked in the select-statement of a cursor, the initial values are inherited from the environment in which the cursor is

opened. For a routine invoked in a nested object (for example a trigger or view), the initial values are inherited from the runtime environment (not inherited from the object definition).

No changes to the special registers are passed back to the invoker of the function.

Non-updatable special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore set to their default values.

external-routine-characteristics

LANGUAGE
This mandatory clause is used to specify the language interface convention to which the user-defined method body is written.

C This means the database manager will call the user-defined method as if it were a C function. The user-defined method must conform to the C language calling and linkage convention as defined by the standard ANSI C prototype.

JAVA
This means the database manager will call the user-defined method as a method in a Java class.

OLE
This means the database manager will call the user-defined method as if it were a method exposed by an OLE automation object. The method must conform with the OLE automation data types and invocation mechanism as described in the *OLE Automation Programmer's Reference*.

LANGUAGE OLE is only supported for user-defined methods stored in Windows 32-bit operating systems. THREADSAFE may not be specified for methods defined with LANGUAGE OLE (SQLSTATE 42613).

PARAMETER STYLE
This clause is used to specify the conventions used for passing parameters to and returning the value from methods.

DB2GENERAL
Used to specify the conventions for passing parameters to and returning the value from external methods that are defined as a method in a Java class. This can only be specified when LANGUAGE JAVA is used.

The value DB2GENRL may be used as a synonym for DB2GENERAL.

SQL
Used to specify the conventions for passing parameters to and returning the value from external methods that conform to C language calling and linkage conventions or methods exposed by OLE automation objects. This must be specified when either LANGUAGE C or LANGUAGE OLE is used.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the external method. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE
Specifies that character data is in UTF-8, and that graphic data is in UCS-2. If the database is not a Unicode database, PARAMETER CCSID UNICODE cannot be specified (SQLSTATE 56031).

This clause cannot be specified with LANGUAGE OLE (SQLSTATE 42613).

DETERMINISTIC or NOT DETERMINISTIC
This optional clause specifies whether the method always returns the same results for given argument values (DETERMINISTIC) or whether the method depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC method must always return the same result from successive invocations with identical inputs. Optimizations taking advantage of the fact that identical inputs always produce the same results are prevented by specifying NOT DETERMINISTIC. An example of a type that is non-deterministic is one that references special registers, global variables, or non-deterministic functions in a way that affects the result type.

FENCED or NOT FENCED
This clause specifies whether the method is considered "safe" to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED).

If a method is registered as FENCED, the database manager protects its internal resources (data buffers, for example) from access by the method. Most methods will have the option of running as FENCED or NOT FENCED. In general, a method running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for methods not adequately checked out can compromise the integrity of a DB2 database. DB2 databases take some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED user-defined methods are used.

Only FENCED can be specified for a method with LANGUAGE OLE or NOT THREADSAFE (SQLSTATE 42613).

If the method is FENCED and has the NO SQL option, the AS LOCATOR clause cannot be specified (SQLSTATE 42613).

Either SYSADM authority, DBADM authority, or a special authority (CREATE_NOT_FENCED_ROUTINE) is required to register a method as NOT FENCED.

THREADSAFE or NOT THREADSAFE
Specifies whether the method is considered “safe” to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the method is defined with LANGUAGE other than OLE:

- If the method is defined as THREADSAFE, the database manager can invoke the method in the same process as other routines. In general, to be threadsafe, a method should not use any global or static data areas.

Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED methods can be THREADSAFE.

- If the method is defined as NOT THREADSAFE, the database manager will never invoke the method in the same process as another routine.

For FENCED methods, THREADSAFE is the default if the LANGUAGE is JAVA. For all other languages, NOT THREADSAFE is the default. If the method is defined with LANGUAGE OLE, THREADSAFE may not be specified (SQLSTATE 42613).

For NOT FENCED methods, THREADSAFE is the default. NOT THREADSAFE cannot be specified (SQLSTATE 42613).

RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT

This optional clause may be used to avoid a call to the external method if any of the non-subject arguments is null.

If RETURNS NULL ON NULL INPUT is specified, and if at execution time any one of the method's arguments is null, the method is not called and the result is the null value.

If CALLED ON NULL INPUT is specified, then regardless of the number of null arguments, the method is called. It can return a null value or a normal (non-null) value. However, responsibility for testing for null argument values lies with the method.

The value NULL CALL may be used as a synonym for CALLED ON NULL INPUT for backwards and family compatibility. Similarly, NOT NULL CALL may be used as a synonym for RETURNS NULL ON NULL INPUT.

There are two cases in which this specification is ignored:

- If the subject argument is null, in which case the method is not executed and the result is null
- If the method is defined to have no parameters, in which case this null argument condition cannot occur.

NO SQL, CONTAINS SQL, READS SQL DATA

Indicates whether the method issues any SQL statements and, if so, what type.

NO SQL

Indicates that the method cannot execute any SQL statements (SQLSTATE 38001).

CONTAINS SQL

Indicates that SQL statements that neither read nor modify SQL data can be executed by the method (SQLSTATE 38004 or 42985). Statements that are not supported in any method return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA

Indicates that some SQL statements that do not modify SQL data can be included in the method (SQLSTATE 38002 or 42985). Statements that are not supported in any method return a different error (SQLSTATE 38003 or 42985).

EXTERNAL ACTION or NO EXTERNAL ACTION

This optional clause specifies whether or not the method takes some action that changes the state of an object not managed by the database manager.

Optimizations that assume methods have no external impacts are prevented by specifying EXTERNAL ACTION.

NO SCRATCHPAD or SCRATCHPAD *length*
This optional clause may be used to specify whether a scratchpad is to be provided for an external method. It is strongly recommended that methods be re-entrant, so a scratchpad provides a means for the method to "save state" from one call to the next.

If SCRATCHPAD is specified, then at the first invocation of the user-defined method, memory is allocated for a scratchpad to be used by the external method. This scratchpad has the following characteristics:

- *length*, if specified, sets the size in bytes of the scratchpad and must be between 1 and 32 767 (SQLSTATE 42820). The default value is 100.
- It is initialized to all X'00''s.
- Its scope is the SQL statement. There is one scratchpad per reference to the external method in the SQL statement.

So, if method X in the following statement is defined with the SCRATCHPAD keyword, three scratchpads would be assigned.

```
SELECT A, X..(A) FROM TABLEB
   WHERE X..(A) > 103 OR X..(A) < 19
```

If ALLOW PARALLEL is specified or defaulted to, then the scope is different from the one shown previously. If the method is executed on multiple database partitions, a scratchpad would be assigned on each database partition where the method is processed, for each reference to the method in the SQL statement. Similarly, if the query is executed with intrapartition parallelism enabled, more than three scratchpads may be assigned.

The scratchpad is persistent. Its content is preserved from one external method call to the next. Any changes made to the scratchpad by the external method on one call will be present on the next call. The database manager initializes scratchpads at the beginning of execution of each SQL statement. The database manager may reset scratchpads at the beginning of execution of each subquery. The system issues a final call before resetting a scratchpad if the FINAL CALL option is specified.

The scratchpad can be used as a central point for system resources (memory, for example) which the external method might acquire. The method could acquire the memory on the first call, keep its address in the scratchpad, and refer to it in subsequent calls.

In such a case where system resource is acquired, the FINAL CALL keyword should also be specified; this causes a special call to be made at end-of-statement to allow the external method to free any system resources acquired.

If SCRATCHPAD is specified, then on each invocation of the user-defined method, an additional argument is passed to the external method which addresses the scratchpad.

If NO SCRATCHPAD is specified, then no scratchpad is allocated or passed to the external method.

NO FINAL CALL or FINAL CALL
This optional clause specifies whether a final call is to be made to an external method. The purpose of such a final call is to enable the external method to free any system resources it has acquired. It can be useful in

conjunction with the SCRATCHPAD keyword in situations where the external method acquires system resources such as memory and anchors them in the scratchpad.

If FINAL CALL is specified, then at execution time, an additional argument is passed to the external method which specifies the type of call. The types of calls are:

- Normal call: SQL arguments are passed and a result is expected to be returned.
- First call: the first call to the external method for this specific reference to the method in this specific SQL statement. The first call is a normal call.
- Final call: a final call to the external method to enable the method to free up resources. The final call is not a normal call. This final call occurs at the following times:
  - End-of-statement: this case occurs when the cursor is closed for cursor-oriented statements, or when the statement is through executing otherwise.
  - End-of-transaction: This case occurs when the normal end-of-statement does not occur. For example, the logic of an application may for some reason bypass the close of the cursor.

  If a commit operation occurs while a cursor defined as WITH HOLD is open, a final call is made at the subsequent close of the cursor or at the end of the application.

If NO FINAL CALL is specified, then no "call type" argument is passed to the external method, and no final call is made.

ALLOW PARALLEL or DISALLOW PARALLEL

This optional clause specifies whether, for a single reference to the method, the invocation of the method can be parallelized. In general, the invocations of most scalar methods should be parallelizable, but there may be methods (such as those depending on a single copy of a scratchpad) that cannot. If either ALLOW PARALLEL or DISALLOW PARALLEL are specified for a method, then DB2 will accept this specification.

The following questions should be considered in determining which keyword is appropriate for the method:.

- Are all the method invocations completely independent of each other? If YES, then specify ALLOW PARALLEL.
- Does each method invocation update the scratchpad, providing value(s) that are of interest to the next invocation (the incrementing of a counter, for example)? If YES, then specify DISALLOW PARALLEL or accept the default.
- Is there some external action performed by the method which should happen only on one database partition? If YES, then specify DISALLOW PARALLEL or accept the default.
- Is the scratchpad used, but only so that some expensive initialization processing can be performed a minimal number of times? If YES, then specify ALLOW PARALLEL.

In any case, the body of every external method should be in a directory that is available on every database partition.

The syntax diagram indicates that the default value is ALLOW PARALLEL. However, the default is DISALLOW PARALLEL if one or more of the following options is specified in the statement:

- NOT DETERMINISTIC
- EXTERNAL ACTION
- SCRATCHPAD
- FINAL CALL

NO DBINFO or DBINFO

This optional clause specifies whether certain specific information known by DB2 will be passed to the method as an additional invocation-time argument (DBINFO), or not (NO DBINFO). NO DBINFO is the default. DBINFO is not supported for LANGUAGE OLE (SQLSTATE 42613). If the method being defined overrides another method, this clause cannot be specified (SQLSTATE 428FV).

If DBINFO is specified, a structure that contains the following information is passed to the method:

- Database name - the name of the currently connected database.
- Application ID - unique application ID which is established for each connection to the database.
- Application Authorization ID - the application runtime authorization ID, regardless of the nested methods in between this method and the application.
- Code page - identifies the database code page.
- Schema name - under the exact same conditions as for Table name, contains the name of the schema; otherwise blank.
- Table name - if and only if the method reference is either the right side of a SET clause in an UPDATE statement, or an item in the VALUES list of an INSERT statement, contains the unqualified name of the table being updated or inserted; otherwise blank.
- Column name - under the exact same conditions as for Table name, contains the name of the column being updated or inserted; otherwise blank.
- Database version/release - identifies the version, release and modification level of the database server invoking the method.
- Platform - contains the server's platform type.
- Table method result column numbers - not applicable to methods.

INHERIT SPECIAL REGISTERS

This optional clause specifies that special registers in the method will inherit their initial values from the calling statement. For cursors, the initial values are inherited from the time that the cursor is opened.

No changes to the special registers are passed back to the caller of the method.

Some special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore never inherited from the caller.

Notes

- Creating a structured type with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of

the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

- A structured subtype defined with no attributes defines a subtype that inherits all its attributes from the supertype. If neither an UNDER clause nor any other attribute is specified, then the type is a root type of a type hierarchy without any attributes.
- The addition of a new subtype to a type hierarchy may cause packages to be invalidated. A package may be invalidated if it depends on a supertype of the new type. Such a dependency is the result of the use of a TYPE predicate or a TREAT specification.
- A structured type may have no more than 4082 attributes (SQLSTATE 54050).
- A method specification is not allowed to have the same signature as a function (comparing the first parameter-type of the function with the subject-type of the method).
- No original method may override another method, or be overridden by an original method (SQLSTATE 42745). Furthermore, a function and a method cannot be in an overriding relationship. This means that if the function were considered to be a method with its first parameter as subject S, it must not override another method in any supertype of S, and it must not be overridden by another method in any subtype of S (SQLSTATE 42745).
- Creation of a structured type automatically generates a set of functions and methods for use with the type. All the functions and methods are generated in the same schema as the structured type. If the signature of the generated function or method conflicts with or overrides the signature of an existing function in this schema, the statement fails (SQLSTATE 42710). The generated functions or methods cannot be dropped without dropping the structured type (SQLSTATE 42917). The following functions and methods are generated:
  - Functions
    - Reference Comparisons

      Six comparison functions with names =, <>, <, <=, >, >= are generated for the reference type REF(*type-name*). Each of these functions takes two parameters of type REF(*type-name*) and returns true, false, or unknown. The comparison operators for REF(*type-name*) are defined to have the same behavior as the comparison operators for the underlying data type of REF(*type-name*). (All references in a type hierarchy have the same reference representation type. This enables REF(S) and REF(T) to be compared, provided that S and T have a common supertype. Because uniqueness of the OID column is enforced only within a table hierarchy, it is possible that a value of REF(T) in one table hierarchy may be "equal" to a value of REF(T) in another table hierarchy, even though they reference different rows.)

      The scope of the reference type is not considered in the comparison.
    - Cast functions

      Two cast functions are generated to cast between the generated reference type REF(*type-name*) and the underlying data type of this reference type.
      - The name of the function to cast from the underlying type to the reference type is the implicit or explicit *funcname1*.

        The format of this function is:

        ```
        CREATE FUNCTION funcname1 (rep-type)
          RETURNS REF(type-name) ...
        ```
      - The name of the function to cast from the reference type to the underlying type of the reference type is the implicit or explicit *funcname2*.

The format of this function is:

```
CREATE FUNCTION funcname2 ( REF(type-name) )
  RETURNS rep-type ...
```

For some rep-types, there are additional cast functions generated with *funcname1* to handle casting from constants.

- If *rep-type* is SMALLINT, the additional generated cast function has the format:

```
CREATE FUNCTION funcname1 (INTEGER)
  RETURNS REF(type-name)
```

- If *rep-type* is CHAR(n), the additional generated cast function has the format:

```
CREATE FUNCTION funcname1 ( VARCHAR(n))
  RETURNS REF(type-name)
```

- If *rep-type* is GRAPHIC(n), the additional generated cast function has the format:

```
CREATE FUNCTION funcname1 (VARGRAPHIC(n))
  RETURNS REF(type-name)
```

The schema name of the structured type must be included in the SQL path for successful use of these operators and cast functions in SQL statements.

- Constructor function

  The constructor function is generated to allow a new instance of the type to be constructed. This new instance will have null for all attributes of the type, including attributes that are inherited from a supertype.

  The format of the generated constructor function is:

```
CREATE FUNCTION type-name ( )
  RETURNS type-name
 ...
```

  If NOT INSTANTIABLE is specified, no constructor function is generated.

– Methods

  - Observer methods

    An observer method is defined for each attribute of the structured type. For each attribute, the observer method returns the type of the attribute. If the subject is null, the observer method returns a null value of the attribute type.

    For example, the attributes of an instance of the structured type ADDRESS can be observed using `C1..STREET`, `C1..CITY`, `C1..COUNTRY`, and `C1..CODE`.

    The method signature of the generated observer method is as if the following statement had been executed:

```
CREATE TYPE  type-name
    ...
  METHOD attribute-name()
    RETURNS attribute-type
```

    where *type-name* is the structured type name.

  - Mutator methods

    A type-preserving mutator method is defined for each attribute of the structured type. Use mutator methods to change attributes within an instance of a structured type. For each attribute, the mutator method returns a copy of the subject modified by assigning the argument to the named attribute of the copy.

For example, an instance of the structured type ADDRESS can be mutated using `C1..CODE('M3C1H7')`. If the subject is null, the mutator method raises an error (SQLSTATE 2202D).

The method signature of the generated mutator method is as if the following statement had been executed:

```
CREATE TYPE  type-name
        ...
     METHOD attribute-name (attribute-type)
        RETURNS type-name
```

If the attribute data type is SMALLINT, REAL, CHAR, or GRAPHIC, an additional mutator method is generated in order to support mutation using constants:

- If *attribute-type* is SMALLINT, the additional mutator supports an argument of type INTEGER.
- If *attribute-type* is REAL, the additional mutator supports an argument of type DOUBLE.
- If *attribute-type* is CHAR, the additional mutator supports an argument of type VARCHAR.
- If *attribute-type* is GRAPHIC, the additional mutator supports an argument of type VARGRAPHIC.

- If the structured type is used as a column type, the length of an instance of the type can be no more than 1 GB in length at runtime (SQLSTATE 54049).

- When creating a new subtype for an existing structured type (for use as a column type), any transform functions already written in support of existing related structured types should be re-examined and updated as necessary. Whether the new type is in the same hierarchy as a given type, or in the hierarchy of a nested type, it is likely that the existing transform function associated with this type will need to be modified to include some or all of the new attributes introduced by the new subtype. Generally speaking, because it is the set of transform functions associated with a given type (or type hierarchy) that enables UDF and client application access to the structured type, the transform functions should be written to support *all* of the attributes in a given composite hierarchy (that is, including the transitive closure of all subtypes and their nested structured types).

  When a new subtype of an existing type is created, all packages dependent on methods that are defined in supertypes of the type being created, and that are eligible for overriding, are invalidated.
- ***Table access restrictions***: If a method is defined as READS SQL DATA, no statement in the method can access a table that is being modified by the statement which invoked the method (SQLSTATE 57053). For example, suppose the method BONUS() is defined as READS SQL DATA. If the statement UPDATE DEPTINFO SET SALARY = SALARY + EMP..BONUS() is invoked, no SQL statement in the BONUS method can read from the EMPLOYEE table.
- ***Privileges***: The definer of the user-defined type always receives the EXECUTE privilege WITH GRANT OPTION on all methods and functions automatically generated for the structured type. The EXECUTE privilege is not granted on any methods explicitly specified in the CREATE TYPE statement until a method body is defined using the CREATE METHOD statement. The definer of the user-defined type does have the right to drop the method specification using the ALTER TYPE statement. EXECUTE privilege on all methods and functions automatically generated during the CREATE TYPE (structured) statement is granted to PUBLIC.

When an external method is used in an SQL statement, the method definer must have the EXECUTE privilege on any packages used by the method.

- In a partitioned database environment, the use of SQL in external user-defined functions or methods is not supported (SQLSTATE 42997).
- Only routines defined as NO SQL can be used to define an index extension (SQLSTATE 428F8).
- A Java routine defined as NOT FENCED will be invoked as if it had been defined as FENCED THREADSAFE.
- ***EXTERNAL ACTION methods***: If an EXTERNAL ACTION method is invoked in other than the outermost select list, the results are unpredictable since the number of times the method is invoked will vary depending on the access plan used.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NOT VARIANT can be specified in place of DETERMINISTIC
  - VARIANT can be specified in place of NOT DETERMINISTIC
  - NULL CALL can be specified in place of CALLED ON NULL INPUT
  - NOT NULL CALL can be specified in place of RETURNS NULL ON NULL INPUT
  - PARAMETER STYLE DB2SQL can be specified in place of PARAMETER STYLE SQL

  The following syntax is accepted as the default behavior for external methods:
  - ASUTIME NO LIMIT
  - NO COLLID
  - PROGRAM TYPE SUB
  - STAY RESIDENT NO
  - CCSID UNICODE in a Unicode database
  - CCSID ASCII in a non-Unicode database if PARAMETER CCSID UNICODE is not specified

  The following syntax is accepted as the default behavior for SQL methods:
  - CCSID UNICODE in a Unicode database
  - CCSID ASCII in a non-Unicode database

Examples

- *Example 1:* Create a type for department.

```
CREATE TYPE DEPT AS
   (DEPT NAME     VARCHAR(20),
      MAX_EMPS INT)
      REF USING INT
   MODE DB2SQL
```

- *Example 2:* Create a type hierarchy consisting of a type for employees and a subtype for managers.

```
CREATE TYPE EMP AS
  (NAME      VARCHAR(32),
  SERIALNUM INT,
  DEPT      REF(DEPT),
  SALARY    DECIMAL(10,2))
  MODE DB2SQL
```

```
CREATE TYPE MGR UNDER EMP AS
  (BONUS     DECIMAL(10,2))
  MODE DB2SQL
```

- *Example 3:* Create a type hierarchy for addresses. Addresses are intended to be used as types of columns. The inline length is not specified, so DB2 will calculate a default length. Encapsulate within the address type definition an external method that calculates how close this address is to a given input address. Create the method body using the CREATE METHOD statement.

```
CREATE TYPE address_t AS
  (STREET     VARCHAR(30),
  NUMBER     CHAR(15),
  CITY       VARCHAR(30),
  STATE      VARCHAR(10))
  NOT FINAL
  MODE DB2SQL
    METHOD SAMEZIP (addr address_t)
    RETURNS INTEGER
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    NO EXTERNAL ACTION,

    METHOD DISTANCE (address_t)
    RETURNS FLOAT
    LANGUAGE C
    DETERMINISTIC
    PARAMETER STYLE SQL
    NO SQL
    NO EXTERNAL ACTION

CREATE TYPE germany_addr_t UNDER address_t AS
  (FAMILY_NAME VARCHAR(30))
  NOT FINAL
  MODE DB2SQL

CREATE TYPE us_addr_t UNDER address_t AS
  (ZIP VARCHAR(10))
  NOT FINAL
  MODE DB2SQL
```

- *Example 4:* Create a type that has nested structured type attributes.

```
CREATE TYPE PROJECT AS
  (PROJ_NAME  VARCHAR(20),
   PROJ_ID    INTEGER,
   PROJ_MGR   MGR,
   PROJ_LEAD  EMP,
   LOCATION   ADDR_T,
   AVAIL_DATE DATE)
   MODE DB2SQL
```

CREATE TYPE MAPPING

The CREATE TYPE MAPPING statement defines a mapping between data types.

The mapping can be defined between the following data types:

- The data type of a column in a data source table or view that is going to be defined to a federated database
- A corresponding data type that is already defined to the federated database

The mapping can associate the federated database data type with a data type at:

- A specified data source
- A range of data sources; for example, all data sources of a particular type and version

A data type mapping must be created only if an existing one is not adequate.

If multiple type mappings are applicable when creating a nickname or creating a table (transparent DDL), the most recent one is applied.

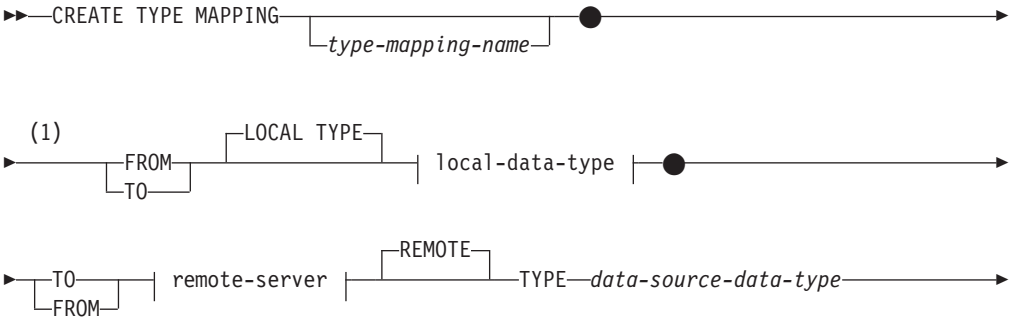
Invocation

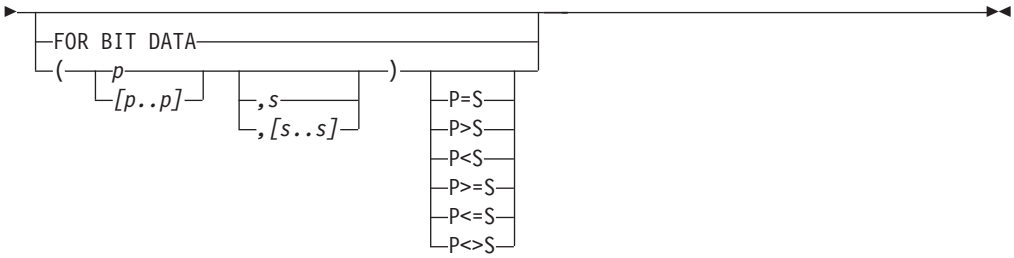
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

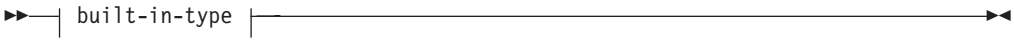
The privileges held by the authorization ID of the statement must include DBADM authority.

Syntax

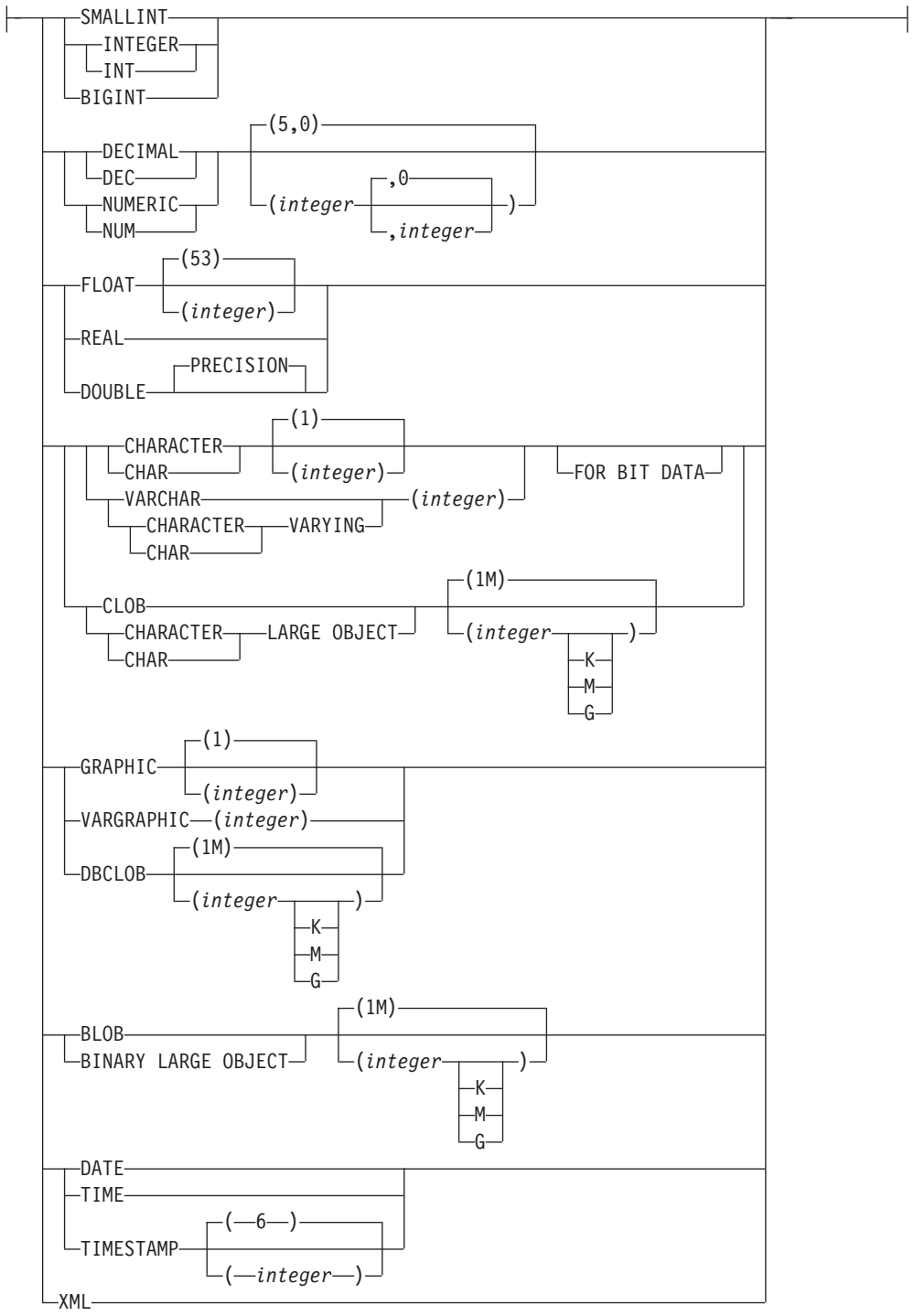




local-data-type:

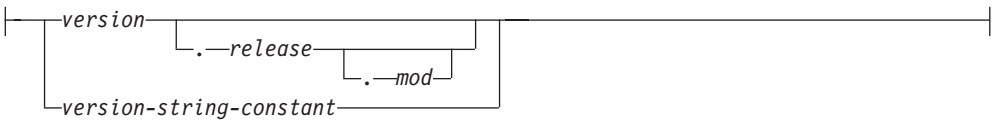


built-in-type:



remote-server:

server-version:



Notes:

1 Both a TO and a FROM keyword must be present in the CREATE TYPE MAPPING statement.

Description

type-mapping-name
Names the data type mapping. The name must not identify a data type mapping that is already described in the catalog. A unique name is generated if *type-mapping-name* is not specified.

FROM or TO
Specifies a reverse or forward type mapping.

FROM
Specifies a forward type mapping when followed by *local-data-type* or a reverse type mapping when followed by *remote-server*.

TO Specifies a forward type mapping when followed by *remote-server* or a reverse type mapping when followed by *local-data-type*.

local-data-type
Identifies a data type that is defined to a federated database. If *local-data-type* is specified without a schema name, the type name is resolved by searching the schemas in the SQL path.

Empty parentheses can be used for the parameterized data types. A parameterized data type is any one of the data types that can be defined with a specific length, scale, or precision. If empty parentheses are specified in a forward type mapping, such as, for example, CHAR(), the length is determined from the column length on the remote table. If empty parentheses are specified in a reverse type mapping, the type mapping is applied to the data type with any length. If you omit parentheses altogether, the default length for the data type is used.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE). NUMBER() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (DECFLOAT or DECIMAL).

DECFLOAT can be accepted only as the *local-data-type* by Oracle wrapper, DB2 wrapper for IBM DB2 Version 9.5 for Linux, UNIX, and Windows or later.

The *local-data-type* cannot be a user-defined type (SQLSTATE 42611).

SERVER *server-name*
Names the data source to which *data-source-data-type* is defined.

SERVER TYPE *server-type*
Identifies the type of data source to which *data-source-data-type* is defined.

VERSION
Identifies the version of the data source to which *data-source-data-type* is defined.

version
Specifies the version number. The value must be an integer.

release
Specifies the number of the release of the version denoted by *version*. The value must be an integer.

mod
Specifies the number of the modification of the release denoted by *release*. The value must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release* and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*
Specifies the name of the wrapper that the federated server uses to interact with data sources of the type and version denoted by *server-type* and *server-version*.

TYPE *data-source-data-type*
Specifies the data source data type that is being mapped to or from the local data type.

Empty parentheses can be used for the parameterized data types. If empty parentheses are specified in a forward type mapping, such as, for example, CHAR(), the type mapping is applied to the data type with any length. If empty parentheses are specified in a reverse type mapping, the length is determined from the column length specified in the transparent DDL. If you omit parentheses altogether, the default length for the data type is used.

The *data-source-data-type* must be a built-in data type. User-defined types are not allowed.

If *server-name* is specified with a type mapping, or existing servers are affected by the type mapping, *data-source-data-type*, *p*, and *s* are verified when creating the type mapping (SQLSTATE 42611).

p If *p* is specified, only the data type whose length or precision equals *p* is affected by the type mapping.

[p1..p2]
For forward type mapping only. For a decimal data type, *p1* and *p2* specify the minimum and maximum number of digits that a value can have. For string data types, *p1* and *p2* specify the minimum and maximum number of characters that a value can have. In all cases, the maximum must equal or exceed the minimum; and both numbers must be valid with respect to the data type.

s If *s* is specified, only the data type whose scale equals *s* is affected by the type mapping.

[s1..s2]
For forward type mapping only. For a decimal data type, *s1* and *s2* specify the minimum and maximum number of digits allowed to the right of the decimal point. The maximum must equal or exceed the minimum, and both numbers must be valid with respect to the data type.

P [operand] S
For a decimal data type, P *[operand]* S specifies a comparison between the

precision and the number of digits allowed to the right of the decimal point. For example, the operand = indicates that the type mapping is applied if the precision and the number of digits allowed in the decimal fraction are the same.

FOR BIT DATA
Indicates whether *data-source-data-type* is for bit data. These keywords are required if the data source type column contains binary values. The database manager will determine this attribute if it is not specified for a character data type.

Notes

- A CREATE TYPE MAPPING statement within a given unit of work (UOW) cannot be processed (SQLSTATE 55007) under either of the following conditions:
  - The statement references a single data source, and the UOW already includes one of the following:
    - A SELECT statement that references a nickname for a table or view within this data source
    - An open cursor on a nickname for a table or view within this data source
    - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within this data source
  - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following:
    - A SELECT statement that references a nickname for a table or view within one of these data sources
    - An open cursor on a nickname for a table or view within one of these data sources
    - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within one of these data sources
- When multiple type mappings are applicable, the most recent one will be used. You can retrieve the creation time for a type mapping by querying the CREATE_TIME column of the SYSCAT.TYPEMAPPINGS catalog view.

Examples

- *Example 1:* Create a forward type mapping between the Oracle data type DATE and the data type SYSIBM.DATE. For all of the nicknames that are created after this mapping is defined, Oracle columns of data type DATE will map to DB2 columns of data type DATE.

```
CREATE TYPE MAPPING MY_ORACLE_DATE
  FROM LOCAL TYPE SYSIBM.DATE
  TO SERVER TYPE ORACLE
  REMOTE TYPE DATE
```

- *Example 2:* Create a forward type mapping between data type SYSIBM.DECIMAL(10,2) and the Oracle data type NUMBER([10..38],2) at data source ORACLE1. If there is a column in the Oracle table of data type NUMBER(11,2), it will be mapped to a column of data type DECIMAL(10,2), because 11 is between 10 and 38.

```
CREATE TYPE MAPPING MY_ORACLE_DEC
  FROM LOCAL TYPE SYSIBM.DECIMAL(10,2)
  TO SERVER ORACLE1
  REMOTE TYPE NUMBER([10..38],2)
```

- *Example 3:* Create a forward type mapping between data type SYSIBM.VARCHAR(*p*) and the Oracle data type CHAR(*p*) at data source

ORACLE1 (*p* is any length). If there is a column in the Oracle table of data type CHAR(10), it will be mapped to a column of data type VARCHAR(10).

```
CREATE TYPE MAPPING MY_ORACLE_CHAR
  FROM LOCAL TYPE SYSIBM.VARCHAR()
  TO SERVER ORACLE1
  REMOTE TYPE CHAR()
```

- *Example 4:* Create a reverse type mapping between the Oracle data type NUMBER(10,2) at data source ORACLE2 and data type SYSIBM.DECIMAL(10,2). If you use transparent DDL to create an Oracle table and specify a column of data type DECIMAL(10,2), DB2 will create the Oracle table with a column of data type NUMBER(10,2).

```
CREATE TYPE MAPPING MY_ORACLE_DEC
  TO LOCAL TYPE SYSIBM.DECIMAL(10,2)
  FROM SERVER ORACLE2
  REMOTE TYPE NUMBER(10,2)
```

CREATE USAGE LIST

The CREATE USAGE LIST statement defines a usage list. A usage list is a database object for monitoring all unique sections (DML statements) that have referenced a particular table or index during their execution.

Invocation

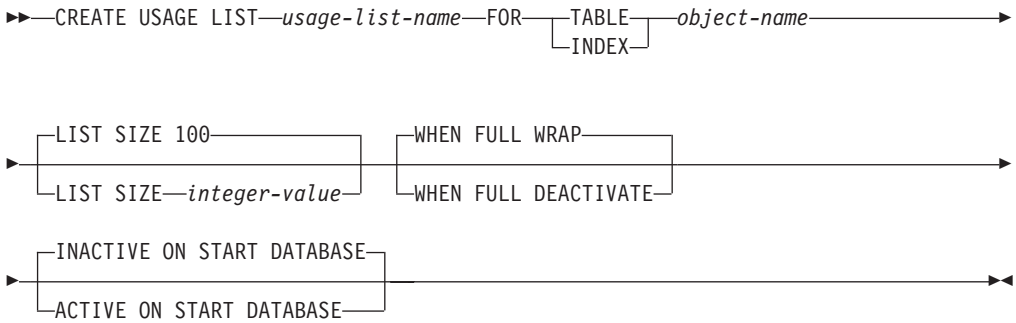
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include one of the following privileges:

- DBADM authority
- SQLADM authority

Syntax



Description

usage-list-name
Names the usage list. The *usage-list-name*, including the implicit or explicit qualifier, must not identify a usage list that is described in the catalog (SQLSTATE 42710). If the usage list is explicitly qualified with a schema name, the schema name must not begin with the characters 'SYS' (SQLSTATE 42939).

TABLE *object-name*
Designates the table for which the usage list is defined. The *object-name*, including the implicit or explicit qualifier, must specify a table defined in the catalog (SQLSTATE 42704). The name must not specify an alias, catalog table, created temporary table, hierarchy table, detached table, nickname, typed table, or view (SQLSTATE 42809).

INDEX *object-name*
Designates the index for which the usage list is defined. The *object-name*, including the implicit or explicit qualifier, must specify an index defined in the catalog (SQLSTATE 42704). Indexes defined on tables other than untyped tables or materialized query tables are not supported (SQLSTATE 42809). The name must specify a physical index; Block Indexes (BLOK), Clustering indexes (CLUS), Dimension block indexes (DIM), Regular indexes (REG), and Physical indexes over XML column (XVIP). All other index types are not supported (SQLSTATE 42809).

LIST SIZE *integer-value*
Specifies that the size of this list is *integer-value* entries. The minimum size that can be specified is 10 and the maximum is 5000 (SQLSTATE 428B7). The default size is 100 entries.

WHEN FULL
Specifies what action is performed when an active usage list becomes full. The default is to wrap when the list becomes full.

WRAP
Specifies that the usage list wraps and replaces the oldest entries.

DEACTIVATE
Specifies that the usage list deactivates.

INACTIVE ON START DATABASE
Specifies that the usage list is not activated for monitoring whenever the database is activated. Collection must be explicitly started using the SET USAGE LIST statement. This clause is the default.

ACTIVE ON START DATABASE
Specifies that the usage list is automatically activated for monitoring whenever the database is activated.

Notes

- ***Tracking sections with unique keys:*** A usage list keep tracks of all unique sections (DML statements only) that have referenced a particular object. References are aggregated within the list with the unique key of executable ID, representing the section doing the reference, and the monitor interval ID at the time of the reference. Each list entry keeps a count of section executions related to that entry and a set of statistics outlining the affect that the section had on the object across those executions.
- ***Usage list release time:*** A usage list is set to released when the CREATE USAGE LIST statement is committed.
- ***Memory allocation:*** Memory is allocated the first time that the object for which the usage list is defined is referenced by a section.
- ***Memory allocation in a partitioned database environment or DB2 pureScale environment:*** If the state of a usage list for a partitioned table or index is set to active, memory is allocated for each data partition when the data partition is first referenced by the section. Similarly, in a partitioned database environment or DB2 pureScale environment, memory is allocated at each active member. If a member is unavailable at the time of activation, then the memory is allocated when the member is next activated (if the state of the usage list is still set to active). This also applies when a member is added to the cluster.
- ***State of the usage list when specifying WHEN FULL DEACTIVATE:*** If the usage list was created with the clause WHEN FULL DEACTIVATE, then the state of the usage list at each member is set to inactive independently. Similarly, for partitioned tables and indexes, the state of the usage list for each data partition is set to inactive independently.
- ***Implicit reactivation of an active usage list:*** If the state of an INACTIVE ON START DATABASE usage list is set to active in a partitioned database environment or DB2 pureScale environment, then its behavior is similar to the ACTIVE ON START DATABASE clause until the state of the usage list is explicitly set to inactive or the instance is recycled. That is, if the state of a usage list is active when a database member is deactivated or offline, and that database member is subsequently reactivated, the usage list for this member is also implicitly reactivated.

- ***Inactive usage lists remain inactive upon database member reactivation:*** If the state of an ACTIVE ON START DATABASE usage list is set to inactive in a partitioned database environment or DB2 pureScale environment, then its behavior is similar to the INACTIVE ON START DATABASE clause until the state of the usage list is explicitly set to active or the instance is recycled. That is, if the state of a usage list is inactive when a database member is deactivated or offline, and that database member is subsequently reactivated, the state of the usage list for this member will remain inactive.
- ***Multiple usage lists:*** Multiple usage lists can be created for the same table or index, however, it is recommended that only one of them be activated. Activating all of them affects database performance and memory usage.
- ***Activating and deactivating usage lists:*** See the Notes section for the SET USAGE LIST STATE statement regarding activation and deactivation of the usage list.
- ***Usage list size considerations:*** When the state of a usage list is set to active, the memory for the usage list is allocated from the monitor heap. At the maximum list size setting, the usage list is approximately 2MB. For partitioned tables or indexes, memory is allocated for each data partition. For example, if a partitioned table has three data partitions defined, approximately 6MB of memory is allocated. Therefore, activating multiple usage lists imposes more memory requirements on the monitor heap. It is therefore suggested that a reasonable list size is selected, or that you set the **mon_heap_sz** configuration parameter to AUTOMATIC so that the database manager manages the monitor heap size.
- ***Performance considerations:*** To maintain high performance, create usage lists such that they are limited to the amount required to gather the information you need. Each usage list requires system memory; system performance can degrade as additional usage lists are activated.

Examples

- *Example 1*: Create a usage list USL_ACC for table SAYYID.ACCOUNTS with a default list size of 100 entries.

```
CREATE USAGE LIST USL_ACC FOR TABLE SAYYID.ACCOUNTS
```

- *Example 2*: Create a usage list USL_SHOPPING_IND for index BIRD.SHOPPINGIND with a list of 50 entries that wraps when the list becomes full.

```
CREATE USAGE LIST USL_SHOPPING_IND FOR INDEX BIRD.SHOPPINGIND
   LISTSIZE 50
   WHEN FULL WRAP
```

- *Example 3*: Create a usage list USL_PAYROLL for table MIKE.PAYROLL with a list size of 200 entries which will deactivate when the list becomes full and will automatically start collecting whenever the database is activated.

```
CREATE USAGE LIST USL_PAYROLL FOR TABLE MIKE.PAYROLL
   LISTSIZE 200
   WHEN FULL DEACTIVATE
   ACTIVE ON START DATABASE
```

- *Example 4*: Create a usage list USL_EMP for partitioned table JACOBO.EMPLOYEES with a list size of 500 entries which will deactivate when the list becomes full.

```
CREATE USAGE LIST USL_EMP FOR TABLE JACOBO.EMPLOYEES
   LIST SIZE 500
   WHEN FULL DEACTIVATE
```

When the usage list is activated for monitoring, then a list of 500 entries will be allocated for each data partition.

- *Example 5*: Create a usage list USL_PARTS for table SHAKTI.PARTS with a list size of 20 entries that will be activated manually on database activation and will wrap when it becomes full.

```
CREATE USAGE LIST USL_PARTS FOR TABLE SHAKTI.PARTS
   LIST SIZE 20
   INACTIVE ON START DATABASE
   WHEN FULL WRAP
```

CREATE USER MAPPING

The CREATE USER MAPPING statement defines a mapping between an authorization ID that uses a federated database and the authorization ID and password to use at a specified data source.

Invocation

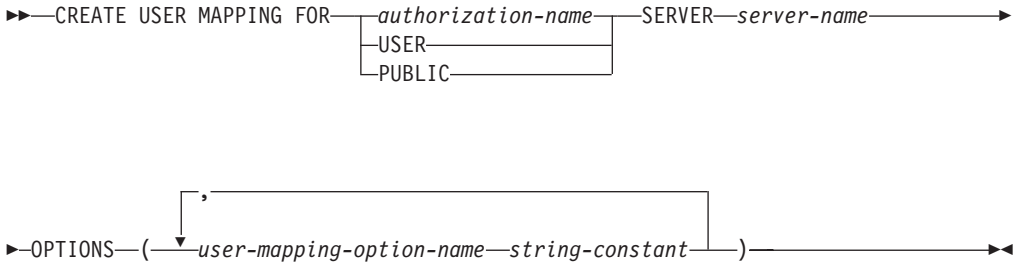
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

If the authorization ID of the statement is different from the authorization name that is being mapped to the data source, the privileges held by the authorization ID of the statement must include DBADM authority. Otherwise, if the authorization ID and the authorization name match, no authorities or privileges are required.

When creating a public user mapping, the privileges held by the authorization ID of the statement must include DBADM authority.

Syntax



Description

authorization-name
Specifies the authorization name under which a user or application connects to a federated database. The *authorization_name* is mapped to the REMOTE_AUTHID user mapping option.

USER
The value in the USER special register. When USER is specified, the authorization ID issuing the CREATE USER MAPPING statement is mapped to the REMOTE_AUTHID user mapping option.

PUBLIC
Specifies that any valid authorization ID for the local federated database will be mapped to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

SERVER *server-name*
Names the server object for the data source that the *authorization-name* can access. The *server-name* is the local name for the remote server that is registered with the federated database.

OPTIONS
Indicates the options that are enabled when the user mapping is created.

user-mapping-option-name
Specifies the name of the option.

string-constant
Specifies the setting for the *user-mapping-option-name* as a character string constant.

Notes

- User mappings are required only for the following data sources: the DB2 family of products, Documentum, Informix, Microsoft SQL Server, ODBC, Oracle, Sybase, and Teradata.
- The REMOTE_PASSWORD option is always required for a user mapping.
- Public user mappings and non-public user mappings cannot coexist on the same federated server. This means that if you have created public user mappings, you will not be able to create non-public user mappings on the same federated server. The reverse is also true, if you have created non-public user mappings, you will not be able to create public user mappings on the same federated server.
- **Syntax alternatives:** The following syntax is supported for compatibility with previous versions of DB2:
  - ADD can be specified before *user-mapping-option-name string-constant*.

Examples

- *Example 1:* Register a user mapping to the DB2 for z/OS data source server object SERVER390. Map the authorization name for the local federated database to the user ID and password for SERVER390. The authorization name is RSPALTEN. The user ID for SERVER390 is SYSTEM. The password for SERVER390 is MANAGER.

```
CREATE USER MAPPING FOR RSPALTEN
  SERVER SERVER390
  OPTIONS
  (REMOTE_AUTHID 'SYSTEM',
  REMOTE_PASSWORD 'MANAGER')
```

- *Example 2:* Register a user mapping to the Oracle data source server object ORACLE1. MARCR is the authorization name for the local federated database and the user ID for ORACLE1. Because the authorization name and the user ID are the same, the REMOTE_AUTHID option does not need to be specified in the user mapping. The password for MARCR on ORACLE1 is NZXCZY .

```
CREATE USER MAPPING FOR MARCR
  SERVER ORACLE1
  OPTIONS
  (REMOTE_PASSWORD 'NZXCZY')
```

- *Example 3:* Create a DRDA wrapper and a DB2 for z/OS data source server SERVER390. Then register a public user mapping to the server object SERVER390. PUBLIC indicates any valid authorization ID for the local federated database. The user ID for SERVER390 is APP_USER. The password for SERVER390 is secret.

```
CREATE WRAPPER DRDA;
CREATE SERVER SERVER390
  TYPE db2/udb VERSION 9.7 WRAPPER DRDA
  AUTHORIZATION "APP_USER" PASSWORD "secret"
  OPTIONS (DBNAME 'remotedb');
CREATE USER MAPPING FOR PUBLIC SERVER SERVER390
  OPTIONS (REMOTE_AUTHID 'APP_USER', REMOTE_PASSWORD 'secret');
```

CREATE VARIABLE

The CREATE VARIABLE statement defines a session global variable.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the variable does not exist
- CREATEIN privilege on the schema, if the schema name of the variable refers to an existing schema
- DBADM authority

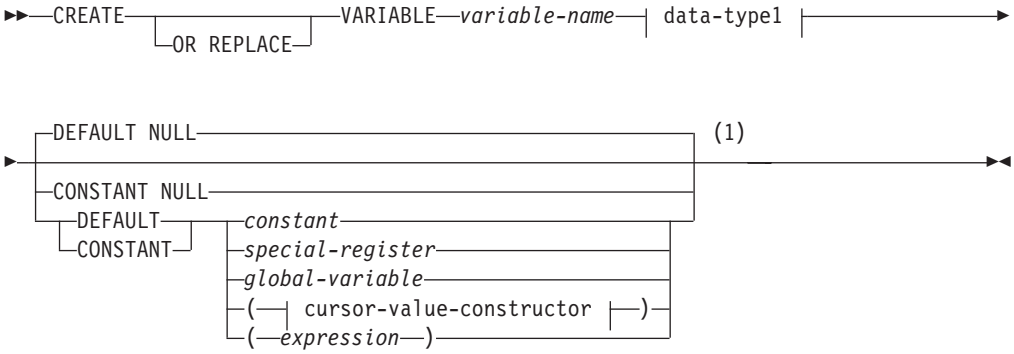
and any privileges that are necessary to execute the default expression.

To execute this statement with a *cursor-value-constructor* that uses a *select-statement*, the privileges held by the authorization ID of the statement must include the privileges necessary to execute the *select-statement*. See the Authorization section in "SQL queries".

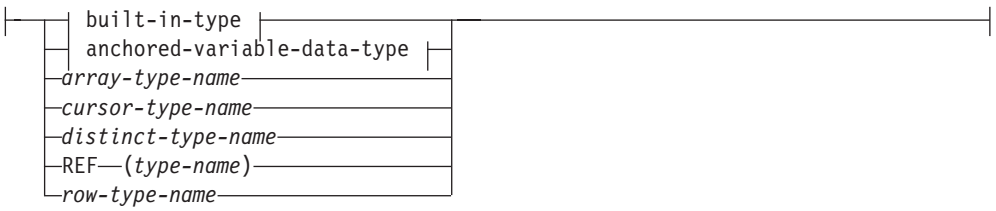
Group privileges are not considered when checking authorization for objects referenced in the statement

To replace an existing variable, the authorization ID of the statement must be the owner of the existing variable (SQLSTATE 42501).

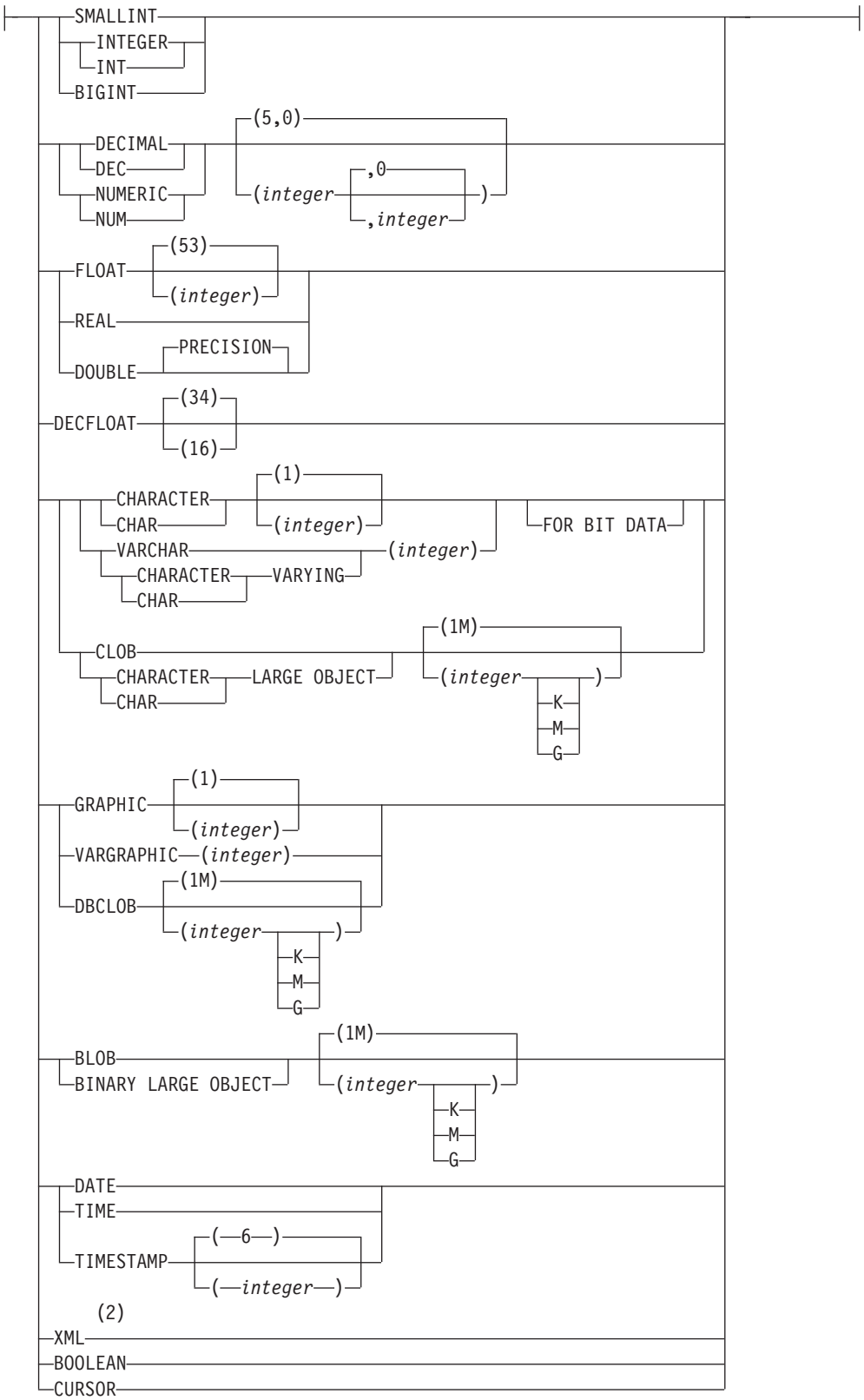
Syntax



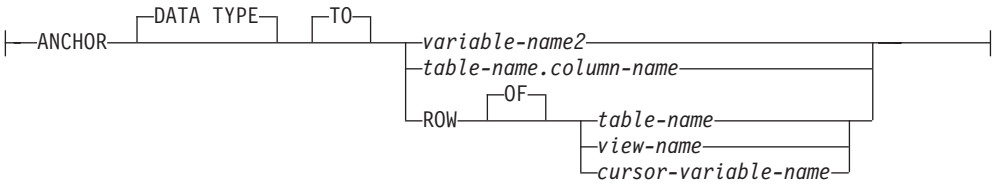
data-type1:



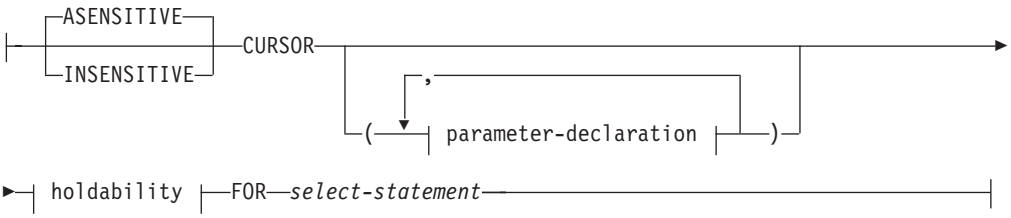
built-in-type:



anchored-variable-data-type:

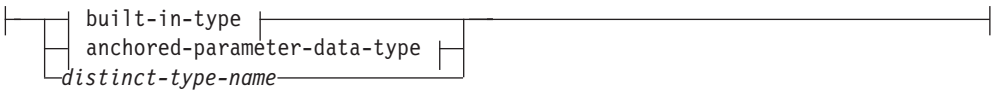


cursor-value-constructor:

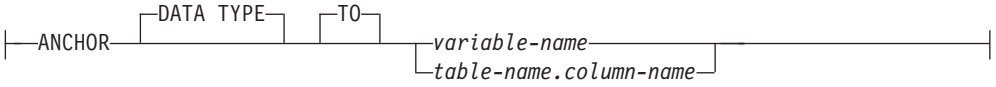


parameter-declaration:

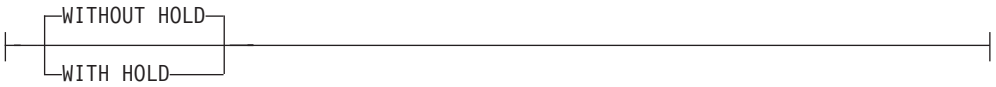
data-type2:



anchored-parameter-data-type:



holdability:



Notes:

1 If *data-type1* specifies a **CURSOR** built-in type or *cursor-type-name*, only **NULL** or *cursor-value-constructor* can be specified. Only **DEFAULT NULL** can be explicitly specified for *array-type-name* or *row-type-name*.

2 For Version 10.1, you can use the XML data type only as a parameter data type in a cursor value constructor. For Version 10.1 Fix Pack 1or later fix pack releases, you can also use the XML data type to create global variables.

Description

OR REPLACE
Specifies to replace the definition for the variable if one exists at the current

server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the variable are not affected. This option is ignored if a definition for the variable does not exist at the current server. This option can be specified only by the owner of the object.

variable-name
Names the global variable. The name, including an implicit or explicit qualifier, must not identify a global variable that already exists at the current server (SQLSTATE 42710). If a qualifier is not specified, the current schema is implicitly assigned. If the global variable name is explicitly qualified with a schema name, the schema name must not begin with the characters 'SYS' (SQLSTATE 42939).

data-type1
Specifies the data type of the global variable. A structured type cannot be specified (SQLSTATE 42611).

built-in-type
Specifies a built-in data type. BOOLEAN and CURSOR cannot be specified for a table. For Version 10.1, an XML data type cannot be specified (SQLSTATE 42611). The XML data type support starts in Version 10.1 Fix Pack 1. For a more complete description of each built-in data type, see "CREATE TABLE".

FOR BIT DATA can be specified as part of character string data types.

BOOLEAN
For a Boolean.

CURSOR
For a reference to an underlying cursor.

anchored-variable-data-type
Identifies another object used to determine the data type of the global variable. The data type of the anchor object has the same limitations that apply to specifying the data type directly, or in the case of a row, to creating a row type.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name2
Identifies a global variable. The data type of the referenced variable is used as the data type for the global variable.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for the global variable.

ROW OF *table-name* **or** *view-name*
Specifies that the global variable is a row of fields with names and data types that are based on the column names and column data types of the table identified by *table-name* or the view identified by *view-name*. The data type of the global variable is an unnamed row type.

ROW OF *cursor-variable-name*
Specifies a row of fields with names and data types that are based on the field names and field data types of the cursor variable identified by *cursor-variable-name*. The specified cursor variable must be one of the following elements (SQLSTATE 428HS):

- A global variable with a strongly typed cursor data type
- A global variable with a weakly typed cursor data type that was created or declared with a CONSTANT clause specifying a select-statement where all the result columns are named.

If the cursor type of the cursor variable is not strongly-typed using a named row type, the data type of the global variable is an unnamed row type.

array-type-name
Specifies the name of a user-defined array type. If *array-type-name* is specified without a schema name, the array type is resolved by searching the schemas in the SQL path.

cursor-type-name
Specifies the name of a cursor type. If *cursor-type-name* is specified without a schema name, the cursor type is resolved by searching the schemas in the SQL path.

distinct-type-name
Specifies the name of a distinct type. The length, precision, and scale of the declared variable are, respectively, the length, precision, and scale of the source type of the distinct type. If *distinct-type-name* is specified without a schema name, the distinct type is resolved by searching the schemas in the SQL path.

REF (*type-name***)**
Specifies a reference type. If a type name is specified without a schema name, the *type-name* is resolved by searching the schemas in the SQL path.

row-type-name
Specifies the name of a user-defined row type. The fields of the variable are the fields of the row type. If *row-type-name* is specified without a schema name, the row type is resolved by searching the schemas in the SQL path.

DEFAULT or CONSTANT
Specifies a value for the global variable when it is first referenced. The DEFAULT or CONSTANT clause value is determined on this first reference. If neither is specified, the default for the global variable is the null value. Only DEFAULT NULL can be explicitly specified if *array-type-name* or *row-type-name* is specified.

DEFAULT
Defines the default for the global variable. The default value must be assignment-compatible with the data type of the variable.

CONSTANT
Specifies that the global variable has a fixed value that cannot be changed. A global variable that is defined using CONSTANT cannot be used as the target of any assignment operation. The fixed value must be assignment-compatible with the data type of the variable.

NULL
Specifies NULL as the default for the global variable. If *row-type-name* is specified, the value for the global variable is a row where each field has the null value.

constant
Specifies the value of a constant as the default for the global variable. If

data-type1 specifies a CURSOR built-in type or *cursor-type-name*, *constant* cannot be specified (SQLSTATE 42601).

special-register
Specifies the value of a special register as the default for the global variable. If *data-type1* specifies a CURSOR built-in type or *cursor-type-name*, *special-register* cannot be specified (SQLSTATE 42601).

global-variable
Specifies the value of a global variable as the default for the global variable. If *data-type1* specifies a CURSOR built-in type or *cursor-type-name*, *global-variable* cannot be specified (SQLSTATE 42601).

cursor-value-constructor
A *cursor-value-constructor* specifies the *select-statement* that is associated with the global variable. The assignment of a *cursor-value-constructor* to a cursor variable defines the underlying cursor of that cursor variable.

ASENSITIVE or INSENSITIVE
Specifies whether the cursor is asensitive or insensitive to changes. See "DECLARE CURSOR" for more information. The default is ASENSITIVE.

ASENSITIVE
Specifies that the cursor should be as sensitive as possible to insert, update, or delete operations made to the rows underlying the result table, depending on how the *select-statement* is optimized. This option is the default.

INSENSITIVE
Specifies that the cursor does not have sensitivity to insert, update, or delete operations that are made to the rows underlying the result table. If INSENSITIVE is specified, the cursor is read-only and the result table is materialized when the cursor is opened. As a result, the size of the result table, the order of the rows, and the values for each row do not change after the cursor is opened. The SELECT statement cannot contain a FOR UPDATE clause, and the cursor cannot be used for positioned updates or deletes.

(parameter-declaration, ...)
Specifies the input parameters of the cursor, including the name and the data type of each parameter.

parameter-name
Names the parameter for use as an SQL variable within *select-statement*. The name cannot be the same as any other parameter name for the cursor. Names should also be chosen to avoid any column names that could be used in *select-statement*, since column names are resolved before parameter names.

data-type2
Specifies the data type of the cursor parameter used within *select-statement*.

built-in-type
Specifies a built-in data type. For a more complete description of each built-in data type, see "CREATE TABLE". The BOOLEAN and CURSOR built-in types cannot be specified (SQLSTATE 429BB).

anchored-parameter-data-type
Identifies another object used to determine the data type of the cursor parameter. The data type of the anchor object is bound by the same limitations that apply when specifying the data type directly.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies a global variable. The data type of the referenced variable is used as the data type for the cursor parameter.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for the cursor parameter.

distinct-type-name
Specifies the name of a distinct type. If *distinct-type-name* is specified without a schema name, the distinct type is resolved by searching the schemas in the SQL path.

holdability
Specifies whether the cursor is prevented from being closed as a consequence of a commit operation. See "DECLARE CURSOR" for more information. The default is WITHOUT HOLD.

WITHOUT HOLD
Does not prevent the cursor from being closed as a consequence of a commit operation.

WITH HOLD
Maintains resources across multiple units of work. Prevents the cursor from being closed as a consequence of a commit operation.

select-statement
Specifies the SELECT statement of the cursor. See "select-statement" for more information.

statement-name
Specifies the prepared *select-statement* of the cursor. See "PREPARE" for an explanation of prepared statements. The target cursor variable must not have a data type that is a strongly-typed user-defined cursor type (SQLSTATE 428HU).

expression
Specifies the value of an expression as the default for the global variable. The expression can be any expression of the type described in "Expressions". The expression must be assignment-compatible with the data type of the variable. The maximum size of the expression is 64K. The default expression must not modify SQL data (SQLSTATE 428FL) or perform external action (SQLSTATE 42845). If *data-type1* specifies a CURSOR built-in type or *cursor-type-name*, *expression* cannot be specified (SQLSTATE 42601).

Rules

- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row

definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

Notes

- Session global variables have a session scope. This means that, although they are available to all sessions that are active on the database, their value is private for each session.
- ***Contexts for array, Boolean, cursor, and row global variables:*** Global variables that are array variables, Boolean variables, or row variables can only be used in compound SQL (compiled) statements or SET variable statements. Global variables that are cursor variables can only be used in compound SQL (compiled) statements.
- ***Create with errors:*** If an object referenced in the default expression does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the variable will still be created successfully. The variable will be marked invalid and will be revalidated the next time it is invoked.
- ***Scope of global variable values:*** The values for session global variables persist until they are updated in the current session, the global variable is dropped or altered, or the application session ends. The value is unaffected by COMMIT or ROLLBACK statements. The default value for a global variable can be not deterministic and dependent on when the default value is calculated for the global variable (for example, a reference to the time of day, or a reference to some data stored in a table).

  A technique commonly used, especially for performance, is for an application or product to manage a set of connections and route transactions to an arbitrary connection. In these situations, the non-default value of a global variable or the not deterministic initial default value for a global variable should only be relied on until the end of the transaction. Examples of where this type of situation can occur include applications that: use XA protocols, use connection pooling, use the connection concentrator, and use HADR to achieve failover.
- ***Privileges to use a global variable:*** An attempt to read from or to write to a global variable created by this statement requires that the authorization ID attempting this action hold the appropriate privilege on the global variable. The definer of the variable is implicitly granted all privileges on the variable.
- ***Setting of the default value:*** A created global variable is instantiated to its default value when it is first referenced within its given scope. Note that if a global variable is referenced in a statement, it is instantiated independently of the control flow for that statement.
- ***Using a newly created session global variable:*** If a global variable is created within a session, it cannot be used by other sessions until the unit of work has committed. However, the new global variable can be used within the session that created the variable before the unit of work commits.

Examples

- *Example 1:* Create a session global variable to indicate what printer to use for the session.

```
CREATE VARIABLE MYSCHEMA.MYJOB_PRINTER VARCHAR(30)
  DEFAULT 'Default printer'
```

- *Example 2:* Create a session global variable to indicate the department where an employee works.

```
CREATE VARIABLE SCHEMA1.GV_DEPTNO INTEGER
  DEFAULT ((SELECT DEPTNO FROM HR.EMPLOYEES
    WHERE EMPUSER = SESSION_USER))
```

- *Example 3:* Create a session global variable to indicate the security level of the current user.

```
CREATE VARIABLE SCHEMA2.GV_SECURITY_LEVEL INTEGER
  DEFAULT (GET_SECURITY_LEVEL (SESSION_USER))
```

- *Example 4:* Create a session global variable as a cursor on the STAFF table that returns the names of each employee for the specified job type. Order the results by the department number.

```
CREATE VARIABLE STAFFJOBS CURSOR
  CONSTANT (CURSOR (WHICHJOB CHAR(5))
  FOR SELECT NAME, DEPT FROM STAFF WHERE JOB = WHICHJOB
    ORDER BY DEPT)
```

- *Example 5:* Create a global variable of the XML data type:

```
CREATE VARIABLE MYSCHEMA.CUSTOMER_HISTORY_VAR XML
```

CREATE VIEW

The CREATE VIEW statement defines a view on one or more tables, views or nicknames.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the view does not exist
- CREATEIN privilege on the schema, if the schema name of the view refers to an existing schema
- DBADM authority

and at least one of the following authorities for each table, view, or nickname identified in any fullselect:

- CONTROL privilege on that table, view, or nickname
- SELECT privilege on that table, view, or nickname
- DATAACCESS authority

If creating a subview:

- The authorization ID of the statement must be the same as the definer of the root table of the table hierarchy, or
- The privileges held by the authorization ID must include DBADM authority

and

- The authorization ID of the statement must have SELECT WITH GRANT privilege on the underlying table of the subview, or the superview must not have SELECT privilege granted to any user other than the view definer, or
- ACCESSCTRL authority and one of the following authorities:
  - SELECT privilege on the underlying table of the subview
  - DATAACCESS authority

If WITH ROW MOVEMENT is specified, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

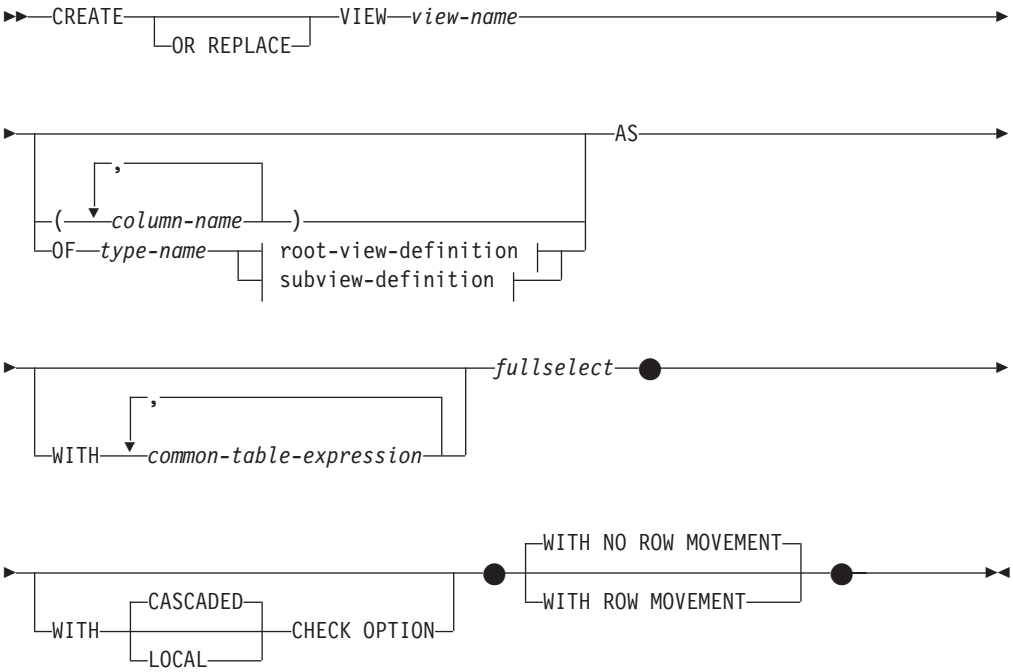
- UPDATE privilege on that table or view
- DATAACCESS authority

Group privileges are not considered for any table or view specified in the CREATE VIEW statement.

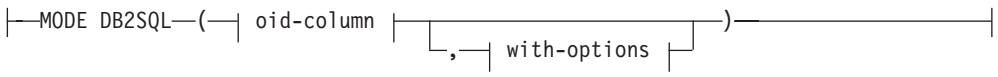
Privileges are not considered when defining a view on a federated database nickname. Authorization requirements of the data source for the table or view referenced by the nickname are applied when the query is processed. The authorization ID of the statement can be mapped to a different remote authorization ID.

To replace an existing view, the authorization ID of the statement must be the owner of the existing view (SQLSTATE 42501).

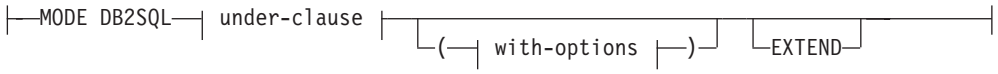
Syntax



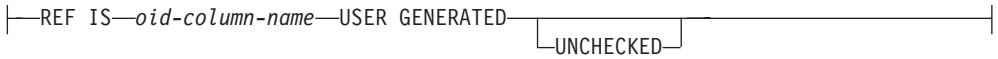
root-view-definition:



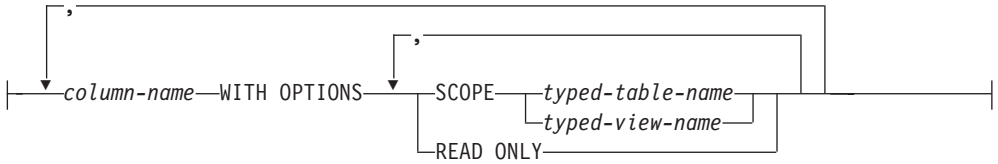
subview-definition:



oid-column:



with-options:



under-clause:

Description

OR REPLACE
Specifies to replace the definition for the view if one exists at the current server. The existing definition is effectively dropped before the new definition is replaced in the catalog, with the exception that privileges that were granted on the view are not affected. This option is ignored if a definition for the view does not exist at the current server. This option can be specified only by the owner of the object.

view-name
Names the view. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname or alias described in the catalog. The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

The name can be the same as the name of an inoperative view (see Inoperative views). In this case the new view specified in the CREATE VIEW statement will replace the inoperative view. The user will get a warning (SQLSTATE 01595) when an inoperative view is replaced. No warning is returned if the application was bound with the bind option SQLWARN set to NO.

column-name
Names the columns in the view. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the view inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the fullselect has duplicate column names or an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

OF *type-name*
Specifies that the columns of the view are based on the attributes of the structured type identified by *type-name*. If *type-name* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be a structured type that is instantiable (SQLSTATE 428DP).

MODE DB2SQL
This clause is used to specify the mode of the typed view. This is the only valid mode currently supported.

UNDER *superview-name*
Indicates that the view is a subview of *superview-name*. The superview must be an existing view (SQLSTATE 42704) and the view must be defined using a structured type that is the immediate supertype of *type-name* (SQLSTATE 428DB). The schema name of *view-name* and *superview-name* must be the same (SQLSTATE 428DQ). The view identified by *superview-name* must not have any existing subview already defined using *type-name* (SQLSTATE 42742).

The columns of the view include the object identifier column of the superview with its type modified to be REF(*type-name*), followed by columns based on the attributes of *type-name* (remember that the type includes the attributes of its supertype).

INHERIT SELECT PRIVILEGES
Any user or group holding a SELECT privilege on the superview will be granted an equivalent privilege on the newly created subview. The subview definer is considered to be the grantor of this privilege.

OID-column
Defines the object identifier column for the typed view.

REF IS *OID-column-name* **USER GENERATED**
Specifies that an object identifier (OID) column is defined in the view as the first column. An OID is required for the root view of a view hierarchy (SQLSTATE 428DX). The view must be a typed view (the OF clause must be present) that is not a subview (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name* (SQLSTATE 42711). The first column specified in *fullselect* must be of type REF(*type-name*) (you may need to cast it so that it has the appropriate type). If UNCHECKED is not specified, it must be based on a not nullable column on which uniqueness is enforced through an index (primary key, unique constraint, unique index, or OID-column). This column will be referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

UNCHECKED
Defines the object identifier column of the typed view definition to assume uniqueness even though the system can not prove this uniqueness. This is intended for use with tables or views that are being defined into a typed view hierarchy where the user knows that the data conforms to this uniqueness rule but it does not comply with the rules that allow the system to prove uniqueness. UNCHECKED option is mandatory for view hierarchies that range over multiple hierarchies or legacy tables or views By specifying UNCHECKED, the user takes responsibility for ensuring that each row of the view has a unique OID. If the user fails to ensure this property, and a view contains duplicate OID values, then a path-expression or DEREF operator involving one of the non-unique OID values may result in an error (SQLSTATE 21000).

with-options
Defines additional options that apply to columns of a typed view.

column-name **WITH OPTIONS**
Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of an attribute defined in (not inherited by) the *type-name* of the view. The column must be a reference type (SQLSTATE 42842). It cannot correspond to a column that also exists in the superview (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS SCOPE clause in the statement (SQLSTATE 42613).

SCOPE
Identifies the scope of the reference type column. A scope must be

specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the DEREF function.

Specifying the scope for a reference type column may be deferred to a subsequent ALTER VIEW statement (if the scope is not inherited) to allow the target table or view to be defined, usually in the case of mutually referencing views and tables. If no scope is specified for a reference type column of the view and the underlying table or view column was scoped, then the underlying column's scope is inherited by the reference type column. The column remains unscoped if the underlying table or view column did not have a scope. See "Notes" on page 886 for more information about scope and reference type columns.

typed-table-name
The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

READ ONLY
Identifies the column as a read-only column. This option is used to force a column to be read-only so that subview definitions can specify an expression for the same column that is implicitly read-only.

AS Identifies the view definition.

WITH *common-table-expression*
Defines a common table expression for use with the fullselect that follows. A common table expression cannot be specified when defining a typed view.

fullselect
Defines the view. At any time, the view consists of the rows that would result if the SELECT statement were executed. The data type of the columns of the view cannot be a distinct type with data type constraints, array type, cursor type, or row type. The fullselect must not reference host variables, parameter markers, or declared temporary tables. However, a parameterized view can be created as an SQL table function.

The fullselect cannot include an SQL data change statement in the FROM clause (SQLSTATE 428FL).

For Typed Views and Subviews: The *fullselect* must conform to the following rules otherwise an error is returned (SQLSTATE 428EA unless otherwise specified).

- The fullselect must not include references to the DBPARTITIONNUM or HASHEDVALUE functions, non-deterministic functions, or functions defined to have external action.

- The body of the view must consist of a single subselect, or a UNION ALL of two or more subselects. Let each of the subselects participating directly in the view body be called a *branch* of the view. A view may have one or more branches.
- The FROM-clause of each branch must consist of a single table or view (not necessarily typed), called the *underlying* table or view of that branch.
- The underlying table or view of each branch must be in a separate hierarchy (that is, a view cannot have multiple branches with their underlying tables or views in the same hierarchy).
- None of the branches of a typed view definition may specify GROUP BY or HAVING.
- If the view body contains UNION ALL, the root view in the hierarchy must specify the UNCHECKED option for its OID column.

For a hierarchy of views and subviews: Let BR1 and BR2 be any branches that appear in the definitions of views in the hierarchy. Let T1 be the underlying table or view of BR1, and let T2 be the underlying table or view of BR2. Then:

- If T1 and T2 are not in the same hierarchy, then the root view in the view hierarchy must specify the UNCHECKED option for its OID column.
- If T1 and T2 are in the same hierarchy, then BR1 and BR2 must contain predicates or ONLY-clauses that are sufficient to guarantee that their row-sets are disjoint.

For typed subviews defined using EXTEND AS: For every branch in the body of the subview:

- The underlying table of each branch must be a (not necessarily proper) subtable of some underlying table of the immediate superview.
- The expressions in the SELECT list must be assignable to the non-inherited columns of the subview (SQLSTATE 42854).

For typed subviews defined using AS without EXTEND:

- For every branch in the body of the subview, the expressions in the SELECT-list must be assignable to the declared types of the inherited and non-inherited columns of the subview (SQLSTATE 42854).
- The OID-expression of each branch over a given hierarchy in the subview must be equivalent (except for casting) to the OID-expression in the branch over the same hierarchy in the root view.
- The expression for a column not defined (implicitly or explicitly) as READ ONLY in a superview must be equivalent in all branches over the same underlying hierarchy in its subviews.

WITH CHECK OPTION

Specifies the constraint that every row that is inserted or updated through the view must conform to the definition of the view. A row that does not conform to the definition of the view is a row that does not satisfy the search conditions of the view.

WITH CHECK OPTION must not be specified if any of the following conditions is true:

- The view is read-only (SQLSTATE 42813). If WITH CHECK OPTION is specified for an updatable view that does not allow inserts, the constraint applies to updates only.
- The view references the DBPARTITIONNUM or HASHEDVALUE function, a non-deterministic function, or a function with external action (SQLSTATE 42997).

- A nickname is the update target of the view.
- A view that has an INSTEAD OF trigger defined on it is the update target of the view (SQLSTATE 428FQ).

If WITH CHECK OPTION is omitted, the definition of the view is not used in the checking of any insert or update operations that use the view. Some checking might still occur during insert or update operations if the view is directly or indirectly dependent on another view that includes WITH CHECK OPTION. Because the definition of the view is not used, rows might be inserted or updated through the view that do not conform to the definition of the view.

CASCADED
The WITH CASCADED CHECK OPTION constraint on a view *V* means that *V* inherits the search conditions as constraints from any updatable view on which *V* is dependent. Furthermore, every updatable view that is dependent on *V* is also subject to these constraints. Thus, the search conditions of *V* and each view on which *V* is dependent are ANDed together to form a constraint that is applied for an insert or update of *V* or of any view dependent on *V*.

LOCAL
The WITH LOCAL CHECK OPTION constraint on a view *V* means the search condition of *V* is applied as a constraint for an insert or update of *V* or of any view that is dependent on *V*.

The difference between CASCADED and LOCAL is shown in the following example. Consider the following updatable views (substituting for Y from column headings of the table that follows):

```
V1 defined on table T
V2 defined on V1 WITH Y CHECK OPTION
V3 defined on V2
V4 defined on V3 WITH Y CHECK OPTION
V5 defined on V4
```

The following table shows the search conditions against which inserted or updated rows are checked:

	Y is LOCAL	**Y is CASCADED**
V1 checked against:	no view	no view
V2 checked against:	V2	V2, V1
V3 checked against:	V2	V2, V1
V4 checked against:	V2, V4	V4, V3, V2, V1
V5 checked against:	V2, V4	V4, V3, V2, V1

Consider the following updatable view which shows the impact of the WITH CHECK OPTION using the default CASCADED option:

```
CREATE VIEW V1 AS SELECT COL1 FROM T1 WHERE COL1 > 10

CREATE VIEW V2 AS SELECT COL1 FROM V1 WITH CHECK OPTION

CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100
```

The following INSERT statement using *V1* will succeed because *V1* does not have a WITH CHECK OPTION and *V1* is not dependent on any other view that has a WITH CHECK OPTION.

```
INSERT INTO V1 VALUES(5)
```

The following INSERT statement using *V2* will result in an error because *V2* has a WITH CHECK OPTION and the insert would produce a row that did not conform to the definition of *V2*.

```
INSERT INTO V2 VALUES(5)
```

The following INSERT statement using *V3* will result in an error even though it does not have WITH CHECK OPTION because *V3* is dependent on *V2* which does have a WITH CHECK OPTION (SQLSTATE 44000).

```
INSERT INTO V3 VALUES(5)
```

The following INSERT statement using *V3* will succeed even though it does not conform to the definition of *V3* (*V3* does not have a WITH CHECK OPTION); it does conform to the definition of *V2* which does have a WITH CHECK OPTION.

```
INSERT INTO V3 VALUES(200)
```

WITH NO ROW MOVEMENT or WITH ROW MOVEMENT
Specifies the action to take for an updatable UNION ALL view when a row is updated in a way that violates a check constraint on the underlying table. The default is WITH NO ROW MOVEMENT.

WITH NO ROW MOVEMENT
Specifies that an error (SQLSTATE 23513) is to be returned if a row is updated in a way that violates a check constraint on the underlying table.

WITH ROW MOVEMENT
Specifies that an updated row is to be moved to the appropriate underlying table, even if it violates a check constraint on that table.

Row movement involves deletion of the rows that violate the check constraint, and insertion of those rows back into the view. The WITH ROW MOVEMENT clause can only be specified for UNION ALL views whose columns are all updatable (SQLSTATE 429BJ). If a row is inserted (perhaps after trigger activation) into the same underlying table from which it was deleted, an error is returned (SQLSTATE 23524). A view defined using the WITH ROW MOVEMENT clause must not contain nested UNION ALL operations, except in the outermost fullselect (SQLSTATE 429BJ). A view defined using the WITH ROW MOVEMENT clause, cannot contain any references to a system-period temporal table, application-period temporal table, or bitemporal table.

Notes

- Creating a view with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- View columns inherit the NOT NULL WITH DEFAULT attribute from the base table or view except when columns are derived from an expression. When a row is inserted or updated into an updatable view, it is checked against the constraints (primary key, referential integrity, and check) if any are defined on the base table.
- A new view cannot be created if it uses an inoperative view in its definition. (SQLSTATE 51024).
- If an object referenced in the view body does not exist or is marked invalid, or the definer temporarily doesn't have privileges to access the object, and if the database configuration parameter **auto_reval** is not set to DISABLED, then the

view will still be created successfully. The view will be marked invalid and will be revalidated the next time it is referenced.

- This statement does not support declared temporary tables (SQLSTATE 42995).
- ***Deletable views:*** A view is *deletable* if an INSTEAD OF trigger for the delete operation has been defined for the view, or if all of the following conditions are true:
  - Each FROM clause of the outer fullselect identifies only one base table (with no OUTER clause), deletable view (with no OUTER clause), deletable nested table expression, or deletable common table expression (cannot identify a nickname). Also, any period-specification specified for the base table or deletable view does not reference the SYSTEM_TIME period.
  - The outer fullselect does not include a VALUES clause
  - The outer fullselect does not include a GROUP BY clause or HAVING clause
  - The outer fullselect does not include aggregate functions in the select list
  - The outer fullselect does not include SET operations (UNION, EXCEPT or INTERSECT) with the exception of UNION ALL
  - The base tables in the operands of a UNION ALL must not be the same table and each operand must be deletable
  - The select list of the outer fullselect does not include DISTINCT
- ***Updatable views:*** A column of a view is *updatable* if an INSTEAD OF trigger for the update operation has been defined for the view, or if all of the following conditions are true:
  - The view is deletable (independent of an INSTEAD OF trigger for delete), the column resolves to a column of a base table (not using a dereference operation), and the READ ONLY option is not specified
  - All the corresponding columns of the operands of a UNION ALL have exactly matching data types (including length or precision and scale) and matching default values if the fullselect of the view includes a UNION ALL

  A view is updatable if *any* column of the view is updatable.
- ***Insertable views:***: A view is insertable if an INSTEAD OF trigger for the insert operation has been defined for the view, or at least one column of the view is updatable (independent of an INSTEAD OF trigger for update), and the fullselect of the view does not include UNION ALL.

  A given row can be inserted into a view (including a UNION ALL) if, and only if, it fulfills the check constraints of exactly one of the underlying base tables.

  To insert into a view that includes non-updatable columns, those columns must be omitted from the column list.
- ***Read-only views***: A view is *read-only* if it is *not* deletable, updatable, or insertable.

  The READONLY column in the SYSCAT.VIEWS catalog view indicates if a view is read-only without considering period specifications or INSTEAD OF triggers.
- Common table expressions and nested table expressions follow the same set of rules for determining whether they are deletable, updatable, insertable, or read-only.
- ***Special registers for temporal support:*** The values of the CURRENT TEMPORAL SYSTEM_TIME and CURRENT TEMPORAL BUSINESS_TIME special registers have no impact on the query expression that defines a view while it is being defined. When a view is used in an SQL statement, the values of the CURRENT TEMPORAL SYSTEM_TIME and CURRENT TEMPORAL BUSINESS_TIME special registers for the session processing the SQL statement are applied to the view.

- ***Inoperative views:*** An *inoperative view* is a view that is no longer available for SQL statements. A view becomes inoperative if:
  - A privilege, upon which the view definition is dependent, is revoked.
  - An object such as a table, nickname, alias or function, upon which the view definition is dependent, is dropped.
  - A view, upon which the view definition is dependent, becomes inoperative.
  - A view that is the superview of the view definition (the subview) becomes inoperative.

  In practical terms, an inoperative view is one in which the view definition has been unintentionally dropped. For example, when an alias is dropped, any view defined using that alias is made inoperative. All dependent views also become inoperative and packages dependent on the view are no longer valid.

  Until the inoperative view is explicitly re-created or dropped, a statement using that inoperative view cannot be compiled (SQLSTATE 51024) with the exception of the CREATE ALIAS, CREATE VIEW, DROP VIEW, and COMMENT ON TABLE statements. Until the inoperative view has been explicitly dropped, its qualified name cannot be used to create another table or alias (SQLSTATE 42710).

  An inoperative view may be re-created by issuing a CREATE VIEW statement using the definition text of the inoperative view. This view definition text is stored in the TEXT column of the SYSCAT.VIEWS catalog. When recreating an inoperative view, it is necessary to explicitly grant any privileges required on that view by others, due to the fact that all authorization records on a view are deleted if the view is marked inoperative. Note that there is no need to explicitly drop the inoperative view in order to re-create it. Issuing a CREATE VIEW statement with the same *view-name* as an inoperative view will cause that inoperative view to be replaced, and the CREATE VIEW statement will return a warning (SQLSTATE 01595).

  Inoperative views are indicated by an X in the VALID column of the SYSCAT.VIEWS catalog view and an X in the STATUS column of the SYSCAT.TABLES catalog view.
- ***Privileges***: The definer of a view always receives the SELECT privilege on the view as well as the right to drop the view. The definer of a view will get CONTROL privilege on the view only if the definer has CONTROL privilege on every base table, view, or nickname identified in the fullselect, or if the definer has each of the following authorities:
  - ACCESSCTRL or SECADM
  - DATAACCESS
  - DBADM

  The definer of the view is granted INSERT, UPDATE, column level UPDATE or DELETE privileges on the view if the view is not read-only and the definer has the corresponding privileges on the underlying objects.

  For a view defined WITH ROW MOVEMENT, the definer acquires the UPDATE privilege on the view only if the definer has the UPDATE privilege on all columns of the view, as well as INSERT and DELETE privileges on all underlying tables or views.

  The definer of a view only acquires privileges if the privileges from which they are derived exist at the time the view is created. The definer must have these privileges either directly or because PUBLIC has these privilege. Privileges are not considered when defining a view on a federated server nickname. However, when using a view on a nickname, the user's authorization ID must have valid select privileges on the table or view that the nickname references at the data

source. Otherwise, an error is returned. Privileges held by groups of which the view definer is a member, are not considered.

When a subview is created, the SELECT privileges held on the immediate superview are automatically granted on the subview.

- ***Scope and REF columns***: When selecting a reference type column in the fullselect of a view definition, consider the target type and scope that is required.
  - If the required target type and scope is the same as the underlying table or view, the column can simply be selected.
  - If the scope needs to be changed, use the WITH OPTIONS SCOPE clause to define the required scope table or view.
  - If the target type of the reference needs to be changed, the column must be cast first to the representation type of the reference and then to the new reference type. The scope in this case can be specified in the cast to the reference type or using the WITH OPTIONS SCOPE clause. For example, assume you select column Y defined as REF(TYP1) SCOPE TAB1. You want this to be defined as REF(VTYP1) SCOPE VIEW1. The select list item would be as follows:

    ```
    CAST(CAST(Y AS VARCHAR(16) FOR BIT DATA) AS REF(VTYP1) SCOPE VIEW1)
    ```

- ***Identity columns:*** A column of a view is considered an identity column, if the element of the corresponding column in the fullselect of the view definition is the name of an identity column of a table, or the name of a column of a view which directly or indirectly maps to the name of an identity column of a base table.

  In all other cases, the columns of a view will not get the identity property. For example:
  - the select-list of the view definition includes multiple instances of the name of an identity column (that is, selecting the same column more than once)
  - the view definition involves a join
  - a column in the view definition includes an expression that refers to an identity column
  - the view definition includes a UNION

  When inserting into a view for which the select list of the view definition directly or indirectly includes the name of an identity column of a base table, the same rules apply as if the INSERT statement directly referenced the identity column of the base table.

- ***Federated views:*** A federated view is a view that includes a reference to a nickname somewhere in the fullselect. The presence of such a nickname changes the authorization model used for the view when the view is subsequently referenced in a query.

  When the view is created, no privilege checking is done to determine whether the view definer has access to the underlying data source table or view of a nickname. Privilege checking of references to tables or views at the federated database are handled as usual, requiring the view definer to have at least SELECT privilege on such objects.

  When a federated view is subsequently referenced in a query, the nicknames result in queries against the data source, and the authorization ID that issued the query (or the remote authorization ID to which it maps) must have the necessary privileges to access the data source table or view. The authorization ID that issues the query referencing the federated view is not required to have any additional privileges on tables or views (non-federated) that exist at the federated server.

- ***ROW MOVEMENT, triggers and constraints:*** When a view that is defined using the WITH ROW MOVEMENT clause is updated, the sequence of trigger and constraints operations is as follows:
  1. BEFORE UPDATE triggers are activated for all rows being updated, including rows that will eventually be moved.
  2. The update operation is processed.
  3. Constraints are processed for all updated rows.
  4. AFTER UPDATE triggers (both row-level and statement-level) are activated in creation order, for all rows that satisfy the constraints after the update operation. Because this is an UPDATE statement, all UPDATE statement-level triggers are activated for all underlying tables.
  5. BEFORE DELETE triggers are activated for all rows that did not satisfy the constraints after the update operation (these are the rows that are to be moved).
  6. The delete operation is processed.
  7. Constraints are processed for all deleted rows.
  8. AFTER DELETE triggers (both row-level and statement-level) are activated in creation order, for all deleted rows. Statement-level triggers are activated for only those tables that are involved in the delete operation.
  9. BEFORE INSERT triggers are activated for all rows being inserted (that is, the rows being moved). The new transition tables for the BEFORE INSERT triggers contain the input data provided by the user. Such triggers cannot contain an UPDATE, a DELETE, or an INSERT operation, or invoke any routine containing such operations (SQLSTATE 42987).
  10. The insert operation is processed.
  11. Constraints are processed for all inserted rows.
  12. AFTER INSERT triggers (both row-level and statement-level) are activated in creation order, for all inserted rows. Statement-level triggers are activated for only those tables that are involved in the insert operation.
- ***Nested UNION ALL views:*** A view defined with UNION ALL and based, either directly or indirectly, on a view that is also defined with UNION ALL cannot be updated if either view is defined using the WITH ROW MOVEMENT clause (SQLSTATE 429BK).
- ***Considerations for implicitly hidden columns:*** It is possible that the result table of the fullselect will include a column of the base table that is defined as implicitly hidden. This can occur when the implicitly hidden column is explicitly referenced in the fullselect of the view definition. However, the corresponding column of the view does not inherit the implicitly hidden attribute. Columns of a view cannot be defined as hidden.
- ***Subselect:***: The *isolation-clause* cannot be specified in the *fullselect* (SQLSTATE 42601).
- ***Obfuscation***: The CREATE VIEW statement can be submitted in obfuscated form. In an obfuscated statement, only the view name is readable. The rest of the statement is encoded in such a way that is not readable but can be decoded by the database server. Obfuscated statements can be produced by calling the DBMS_DDL.WRAP function.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products.

– The FEDERATED keyword can be specified between the keywords CREATE and VIEW. The FEDERATED keyword is ignored, however, because a warning is no longer returned if federated objects are used in the view definition.

Examples

- *Example 1:* Create a view named MA_PROJ upon the PROJECT table that contains only those rows with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE VIEW MA_PROJ  AS SELECT *
  FROM PROJECT
   WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

- *Example 2:* Create a view as in example 1, but select only the columns for project number (PROJNO), project name (PROJNAME) and employee in charge of the project (RESPEMP).

```
CREATE VIEW MA_PROJ
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

- *Example 3:* Create a view as in example 2, but, in the view, call the column for the employee in charge of the project IN_CHARGE.

```
CREATE VIEW MA_PROJ
  (PROJNO, PROJNAME, IN_CHARGE)
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

  Note: Even though only one of the column names is being changed, the names of all three columns in the view must be listed in the parentheses that follow MA_PROJ.

- *Example 4:* Create a view named PRJ_LEADER that contains the first four columns (PROJNO, PROJNAME, DEPTNO, RESPEMP) from the PROJECT table together with the last name (LASTNAME) of the person who is responsible for the project (RESPEMP). Obtain the name from the EMPLOYEE table by matching EMPNO in EMPLOYEE to RESPEMP in PROJECT.

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
```

- *Example 5:* Create a view as in example 4, but in addition to the columns PROJNO, PROJNAME, DEPTNO, RESPEMP, and LASTNAME, show the total pay (SALARY + BONUS + COMM) of the employee who is responsible. Also select only those projects with mean staffing (PRSTAFF) greater than one.

```
CREATE VIEW PRJ_LEADER
(PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, TOTAL_PAY )
AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, SALARY+BONUS+COMM
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
  AND PRSTAFF > 1
```

  Specifying the column name list could be avoided by naming the expression SALARY+BONUS+COMM as TOTAL_PAY in the fullselect.

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP,
              LASTNAME, SALARY+BONUS+COMM AS TOTAL_PAY
    FROM PROJECT, EMPLOYEE
    WHERE RESPEMP = EMPNO AND PRSTAFF > 1
```

- *Example 6:* Given the set of tables and views shown in the following figure: User ZORPIE (who does not have ACCESSCTRL, DATAACCESS, or DBADM

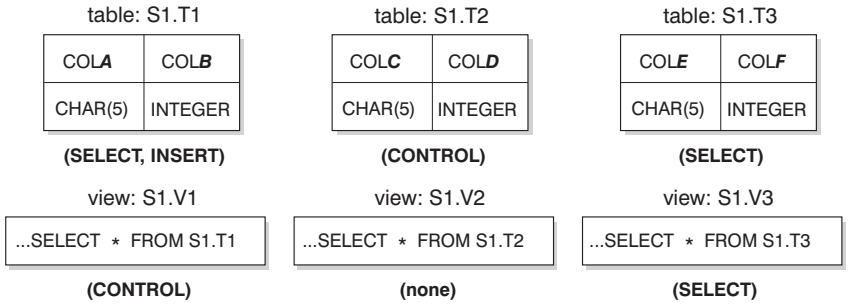


Figure 1. Tables and Views for Example 6

authority) has the privileges shown in parentheses for each object:

1. ZORPIE will get CONTROL privilege on the view that she creates with:

   ```
   CREATE VIEW VA AS SELECT * FROM S1.V1
   ```

   because she has CONTROL on S1.V1. (CONTROL on S1.V1 must have been granted to ZORPIE by someone with ACCESSCTRL or SECADM authority.) It does not matter which, if any, privileges she has on the underlying base table.
2. ZORPIE will not be allowed to create the view:

   ```
   CREATE VIEW VB AS SELECT * FROM S1.V2
   ```

   because she has neither CONTROL nor SELECT on S1.V2. It does not matter that she has CONTROL on the underlying base table (S1.T2).
3. ZORPIE will get CONTROL privilege on the view that she creates with:

   ```
   CREATE VIEW VC (COLA, COLB, COLC, COLD)
     AS SELECT * FROM S1.V1, S1.T2
     WHERE COLA = COLC
   ```

   because the fullselect of ZORPIE.VC references view S1.V1 and table S1.T2 and she has CONTROL on both of these. Note that the view VC is read-only, so ZORPIE does not get INSERT, UPDATE or DELETE privileges.
4. ZORPIE will get SELECT privilege on the view that she creates with:

   ```
   CREATE VIEW VD (COLA,COLB, COLE, COLF)
     AS SELECT * FROM S1.V1, S1.V3
     WHERE COLA = COLE
   ```

   because the fullselect of ZORPIE.VD references the two views S1.V1 and S1.V3, one on which she has only SELECT privilege, and one on which she has CONTROL privilege. She is given the lesser of the two privileges, SELECT, on ZORPIE.VD.
5. ZORPIE will get INSERT, UPDATE and DELETE privilege WITH GRANT OPTION and SELECT privilege on the view VE in the following view definition.

   ```
   CREATE VIEW VE
      AS SELECT * FROM S1.V1
     WHERE COLA > ANY
           (SELECT COLE FROM S1.V3)
   ```

   ZORPIE's privileges on VE are determined primarily by her privileges on S1.V1. Since S1.V3 is only referenced in a subquery, she only needs SELECT privilege on S1.V3 to create the view VE. The definer of a view only gets

CONTROL on the view if they have CONTROL on all objects referenced in the view definition. ZORPIE does not have CONTROL on S1.V3, consequently she does not get CONTROL on VE.

CREATE WORK ACTION SET

The CREATE WORK ACTION SET statement defines a work action set and work actions within the work action set.

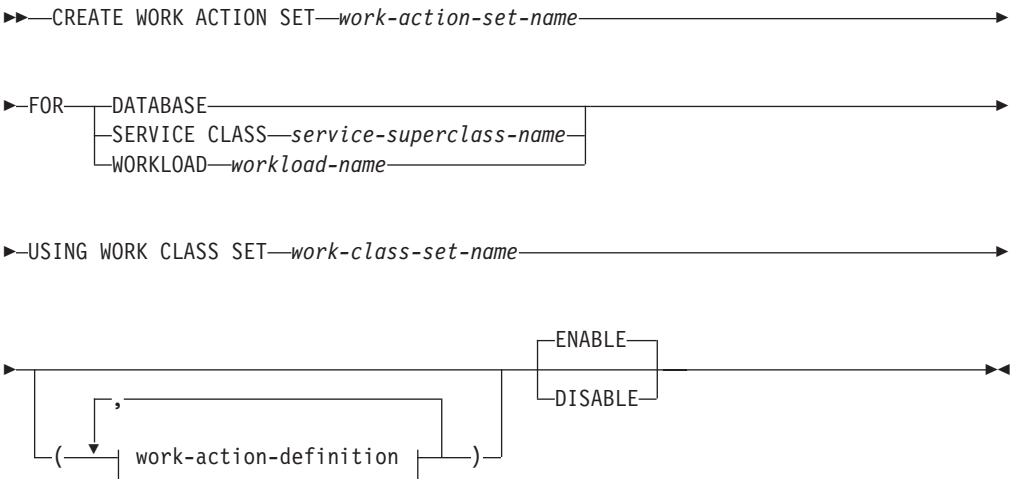
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

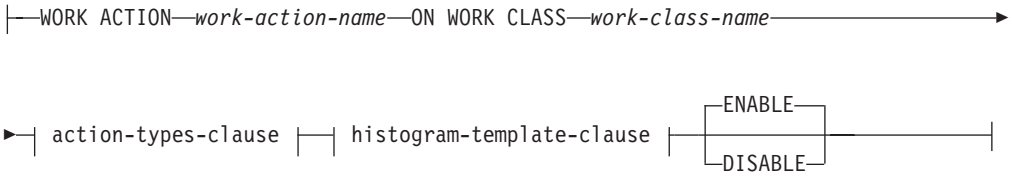
Authorization

The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

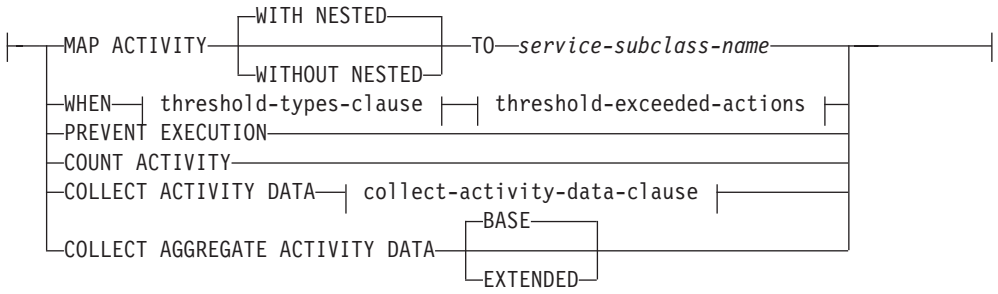
Syntax



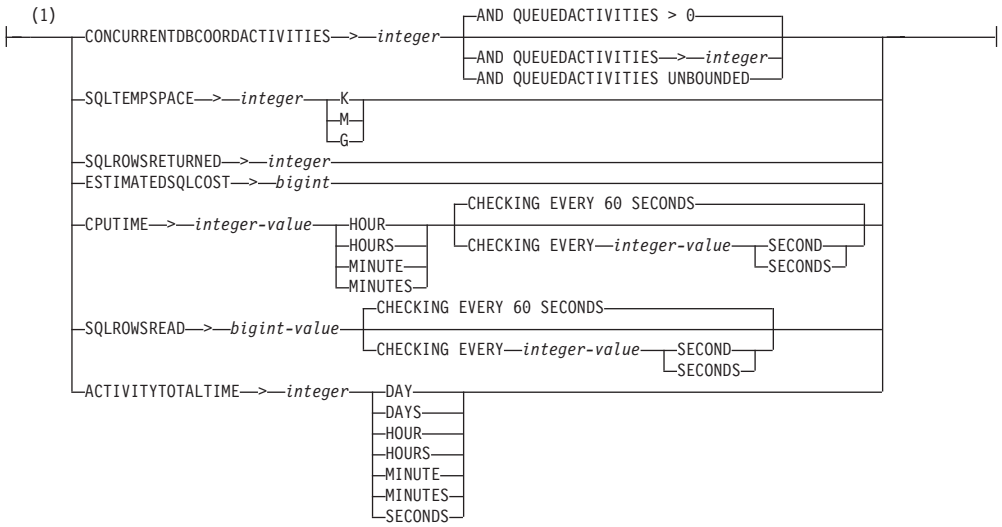
work-action-definition:



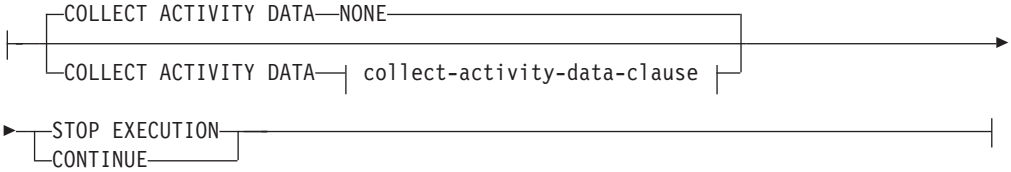
action-types-clause:



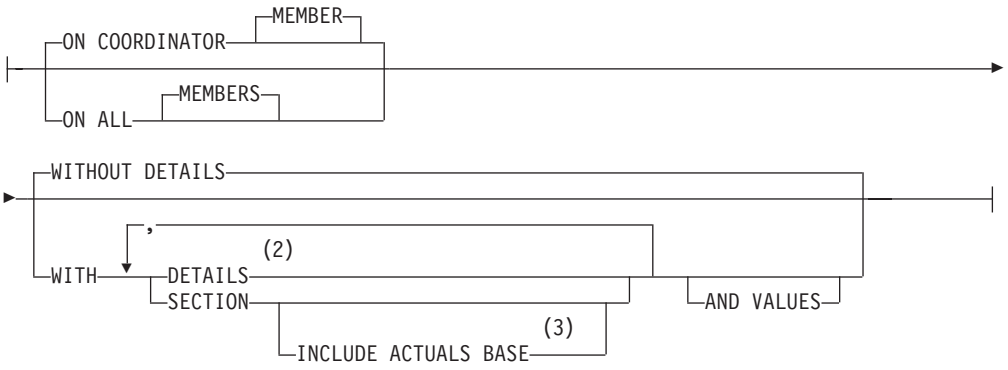
threshold-types-clause:



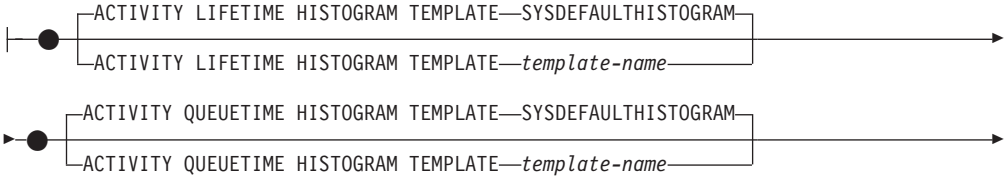
threshold-exceeded-actions:

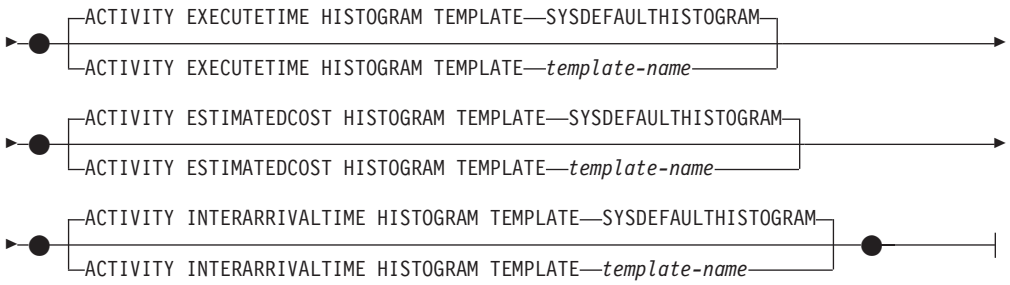


collect-activity-data-clause:



histogram-template-clause:





Notes:

1 Only one work action of the same threshold type can be applied to a single work class at a time.

2 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

3 This clause does not apply to thresholds.

Description

work-action-set-name
Names the work action set. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *work-action-set-name* must not identify a work action set that already exists at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

FOR
Specifies the database manager object to which the actions in this work action set will apply. Each database manager object can have only one work action set defined for it (SQLSTATE 5U017).

DATABASE
The actions in this work action set are to apply to the database. If DATABASE is specified, the MAP ACTIVITY action cannot be specified (SQLSTATE 5U034).

SERVICE CLASS *service-superclass-name*
The actions in this work action set are to apply to *service-superclass-name*. If SERVICE CLASS is specified, threshold actions cannot be specified (SQLSTATE 5U034). The *service-superclass-name* must exist at the current server (SQLSTATE 42704). The *service-superclass-name* must not be a service subclass and cannot be any of the following classes (SQLSTATE 5U032):

- The system service class (SYSDEFAULTSYSTEMCLASS)
- The maintenance service class (SYSDEFAULTMAINTENANCECLASS)
- The default user service class (SYSDEFAULTUSERCLASS)

WORKLOAD *workload-name*
The actions in this work action set are to apply to workload *workload-name*. If WORKLOAD is specified, the MAP ACTIVITY action cannot be specified (SQLSTATE 5U034). The *workload-name* must exist at the current server (SQLSTATE 42704). The *workload-name* cannot be the SYSDEFAULTADMWORKLOAD (SQLSTATE 5U032).

USING WORK CLASS SET *work-class-set-name*
Specifies the work class set containing the work classes that will classify database activities on which to perform actions. The *work-class-set-name* must exist at the current server (SQLSTATE 42704).

work-action-definition
Specifies the definition of the work action.

WORK ACTION *work-action-name*
Names the work action. The *work-action-name* must not identify a work action that already exists at the current server under this work action set (SQLSTATE 42710). The *work-action-name* cannot begin with 'SYS' (SQLSTATE 42939).

ON WORK CLASS *work-class-name*
Specifies the work class that identifies the database activities to which this work action will apply. The *work-class-name* must exist in the *work-class-set-name* at the current server (SQLSTATE 42704).

MAP ACTIVITY
Specifies a work action of mapping the activity. This action can only be specified if the object for which this work action set is defined is a service superclass (SQLSTATE 5U034).

WITH NESTED or WITHOUT NESTED
Specifies whether or not activities that are nested under this activity are mapped to the service subclass. The default is WITH NESTED.

WITH NESTED
All database activities that have a nesting level of zero that are classified under the work class, and all database activities nested under this activity, are mapped to the service subclass; that is, activities with a nesting level greater than zero are run under the same service class as activities with a nesting level of zero.

WITHOUT NESTED
Only database activities that have a nesting level of zero that are classified under the work class are mapped to the service subclass. Database activities that are nested under this activity are handled according to their activity type.

TO *service-subclass-name*
Specifies the service subclass to which activities are to be mapped. The *service-subclass-name* must already exist in the *service-superclass-name* at the current server (SQLSTATE 42704). The *service-subclass-name* cannot be the default service subclass, SYSDEFAULTSUBCLASS (SQLSTATE 5U018).

WHEN
Specifies the threshold that will be applied to the database activity that is associated with the work class for which this work action is defined. A threshold can only be specified if the database manager object for which this work action set is defined is a database or a workload (SQLSTATE 5U034). None of these thresholds apply to internal database activities initiated by the database manager or to database activities generated by administrative SQL routines.

threshold-types-clause
For a description of valid threshold types, see "CREATE THRESHOLD" statement.

threshold-exceeded-actions
For a description of valid threshold-exceeded actions, see "CREATE THRESHOLD" statement.

PREVENT EXECUTION
Specifies that none of the database activities associated with the work class for which this work action is defined will be allowed to run (SQLSTATE 5U033).

COUNT ACTIVITY
Specifies that all of the database activities associated with the work class for which this work action is defined are to be run and that each time one is run, the counter for the work class will be incremented.

COLLECT ACTIVITY DATA
Specifies that data about each activity associated with the work class for which this work action is defined is to be sent to any active activities event monitor when the activity completes. The default is COLLECT ACTIVITY DATA WITHOUT DETAILS.

collect-activity-data-clause

ON COORDINATOR MEMBER
Specifies that the activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that activity data is to be collected at all members where the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. If the AND VALUES clause is specified, activity input values will be collected only for the members of the coordinator.

WITHOUT DETAILS
Specifies that data about each activity that executes in the service class should be sent to any active activities event monitor, when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. Section actuals will be collected on any member where the activity data is collected.

INCLUDE ACTUALS BASE
Specifies that section actuals should also be collected on any partition where the activity data is collected. For section actuals to be collected, either INCLUDE ACTUALS clause must be specified or the **section_actuals** database configuration parameter must be set.

The effective setting for the collection of section actuals is the combination of the INCLUDE ACTUALS clause, the **section_actuals** database configuration parameter, and the <collectsectionactuals> setting specified on the WLM_SET_CONN_ENV routine. For example, if INCLUDE ACTUALS BASE is specified, yet the **section_actuals** database configuration parameter value is NONE and <collectsectionactuals> is set to NONE, then the effective setting for the collection of section actuals is BASE.

BASE specifies that the following should be enabled and collected during the activity's execution:

- Basic operator cardinality counts
- Statistics for each object referenced (DML statements only)

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

COLLECT AGGREGATE ACTIVITY DATA
Specifies that aggregate activity data is to be captured for activities that are associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default is COLLECT AGGREGATE ACTIVITY DATA BASE. This clause cannot be specified for a work action defined in a work action set that is applied to a database.

BASE
Specifies that basic aggregate activity data should be captured for activities associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. Basic aggregate activity data includes:

- Estimated activity cost high watermark
- Rows returned high watermark
- Temporary table space usage high watermark. Only activities that have an SQLTEMPSPACE threshold applied to them participate in this high watermark.
- Activity life time histogram
- Activity queue time histogram
- Activity execution time histogram

EXTENDED
Specifies that all aggregate activity data should be captured for activities associated with the work class for which this work action is defined and sent to the statistics event monitor, if one is active. This includes all basic aggregate activity data plus:

- Activity data manipulation language (DML) estimated cost histogram
- Activity DML inter-arrival time histogram

ENABLE or DISABLE
Specifies whether or not the work action is to be considered when database activities are submitted. The default is ENABLE.

ENABLE
Specifies that the work action is enabled and will be considered when database activities are submitted.

DISABLE
Specifies that the work action is disabled and will not be considered when database activities are submitted.

ENABLE or DISABLE
Specifies whether or not the work action set is to be considered when database activities are submitted. The default is ENABLE.

ENABLE
Specifies that the work action set is enabled and will be considered when database activities are submitted.

DISABLE
Specifies that the work action set is disabled and will not be considered when database activities are submitted.

histogram-template-clause
Specifies histogram templates to use when collecting aggregate activity data for activities associated with the work class to which this work action is assigned. Aggregate activity data is only collected for the work class when the work action type is COLLECT AGGREGATE ACTIVITY DATA.

ACTIVITY LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of DB2 activities—associated with the work class to which this work action is assigned—running during a specific interval. This time includes both time queued and time executing. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY QUEUETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities—associated with the work class to which this work action is assigned—are queued during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities—associated with the work class to which this work action is assigned—are executing during a specific interval. This time does not include the time spent queued. Activity execution time is collected in this histogram at each member where the activity executes. On the activity's coordinator member, this is the end-to-end execution time (that is, the life time less the time spent queued). On non-coordinator members, this is the time that these members spend working on behalf of the activity. During the execution of a given activity, DB2 might present work to a non-coordinator member more than once, and each time the

non-coordinator member will collect the execution time for that occurrence of the activity. Therefore, the counts in the execution time histogram might not represent the actual number of unique activities that executed on a member. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY ESTIMATEDCOST HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the estimated cost, in timerons, of DML activities associated with the work class to which this work action is assigned. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

ACTIVITY INTERARRIVALTIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, between the arrival of one DML activity and the arrival of the next DML activity, for any activity associated with the work class to which this work action is assigned. The default is SYSDEFAULTHISTOGRAM. This information is only collected when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (histogram template)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (service class)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (threshold)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (work action set)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (work class set)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (workload)
  - GRANT (workload privileges) or REVOKE (workload privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.

  - DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
- The enforcement scope is determined automatically based on the threshold type. For CONCURRENTDBCOORDACTIVITIES type thresholds, the environment is also used to determine the enforcement scope where it defaults to the DATABASE enforcement scope in environments other than DB2 pureScale, and the MEMBER enforcement scope in DB2 pureScale environments.

Examples

- *Example 1:* Create a work action set named DATABASE_ACTIONS to apply to all database activities. Use the LARGE_QUERIES work class set and define the following work actions. Work action ONE_CONCURRENT_QUERY has a threshold action that allows one concurrent query to run on the system at a time for queries that fall within the LARGE_ESTIMATED_COST work class. If that threshold is exceeded, the database manager is to queue the activity, but is not to allow more than one database activity to be queued at a time. If the queue threshold is exceeded, the database activity is not to be allowed to run. Work action TWO_CONCURRENT_QUERIES has a threshold action that allows two concurrent queries to execute at the same time for queries that fall within the LARGE_CARDINALITY work class, and allows no more than two to be queued. If more than two queries are to be queued, the database activity is to continue putting the queries in the queue and is to collect the database activity data in the activities event monitor, if one is active.

```
CREATE WORK ACTION SET DATABASE_ACTIONS
  FOR DATABASE USING WORK CLASS SET LARGE_QUERIES
    (WORK ACTION ONE_CONCURRENT_QUERY ON WORK CLASS LARGE_ESTIMATED_COST
     WHEN CONCURRENTDBCOORDACTIVITIES > 1 AND QUEUEDACTIVITIES > 1
       STOP EXECUTION,
     WORK ACTION TWO_CONCURRENT_QUERIES ON WORK CLASS LARGE_CARDINALITY
     WHEN CONCURRENTDBCOORDACTIVITIES > 2 AND QUEUEDACTIVITIES > 2
       COLLECT ACTIVITY DATA CONTINUE)
```

- *Example 2:* Create a work action set named ADMIN_APPS_ACTIONS with one work action named MAP_SELECTS that is to apply to database activities that run under service superclass ADMIN_APPS. The work action is to map all database activity that falls within the SELECT_CLASS work class to service subclass SELECTS_SERVICE_CLASS, which is in the DML_SELECTS work class set.

```
CREATE WORK ACTION SET ADMIN_APPS_ACTIONS
  FOR SERVICE CLASS ADMIN_APPS USING
    WORK CLASS SET DML_SELECTS
      (WORK ACTION MAP_SELECTS ON WORK CLASS SELECT_CLASS
       MAP ACTIVITY TO SELECTS_SERVICE_CLASS)
```

CREATE WORK CLASS SET

The CREATE WORK CLASS SET statement defines a work class set.

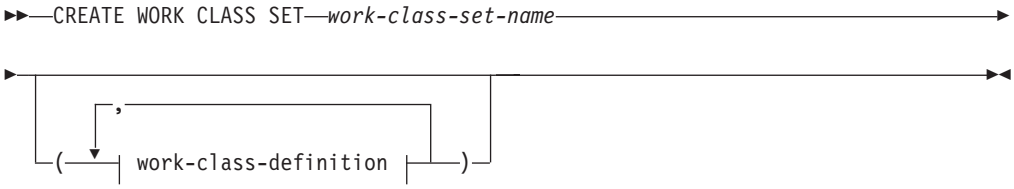
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

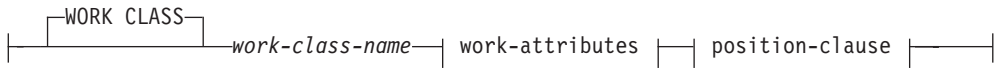
Authorization

The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

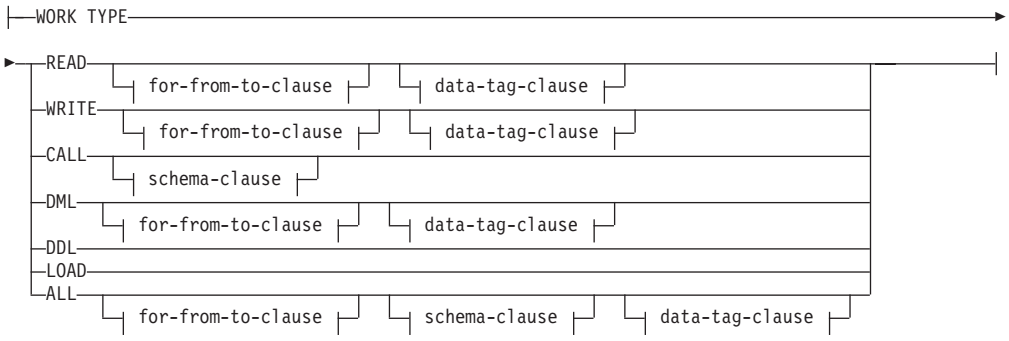
Syntax



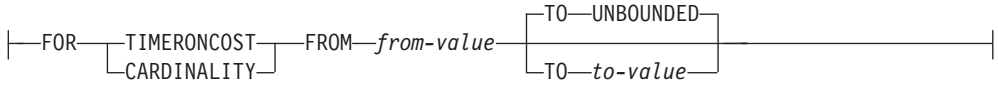
work-class-definition:



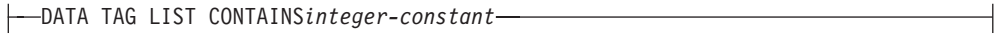
work-attributes:



for-from-to-clause:

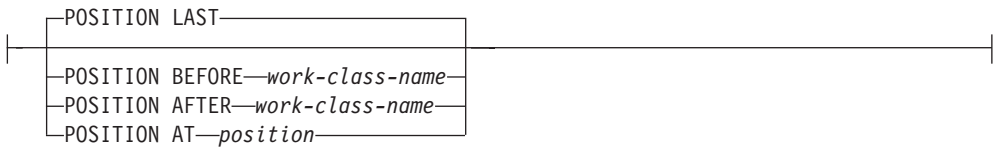


data-tag-clause:



schema-clause:

position-clause:



Description

work-class-set-name
Names the work class set. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *work-class-set-name* must not identify a work class set that already exists at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

work-class-definition
Specifies the definition of the work class.

WORK CLASS *work-class-name*
Names the work class. The *work-class-name* must not identify a work class that already exists within the work class set at the current server (SQLSTATE 42710). The *work-class-name* cannot begin with 'SYS' (SQLSTATE 42939).

work-attributes
The attributes of the database activity must match all of the attributes specified in this work class if that activity is to be associated with this work class.

WORK TYPE
Specifies the type of database activity.

READ
This activity includes the following statements:

- All SELECT or SELECT INTO statements that do not contain a DELETE, INSERT, MERGE, or UPDATE statement, and all VALUES INTO statements
- All XQuery statements

WRITE
This activity includes the following statements:

- UPDATE
- DELETE
- INSERT
- MERGE
- All SELECT statements that contain a DELETE, INSERT, or UPDATE statement, and all VALUES INTO statements

CALL
Includes the CALL statement. A CALL statement is considered for a work class with a work type of CALL or ALL.

DML
Includes the statements listed under READ and WRITE.

DDL
This activity includes the following statements:
- ALTER
- CREATE
- COMMENT
- DECLARE GLOBAL TEMPORARY TABLE
- DROP
- FLUSH PACKAGE CACHE
- GRANT
- REFRESH TABLE
- RENAME
- REVOKE
- SET INTEGRITY

LOAD
DB2 load operations.

ALL
All recognized workload management (WLM) activity that falls under any one of the keywords previously listed within the description for WORK TYPE.

FOR
Indicates the type of information that is being specified in the FROM *from-value* TO *to-value* clause. The FOR clause is only used for the following work types:
- ALL
- DML
- READ
- WRITE

TIMERONCOST
The estimated cost of the work, in timerons. This value is used to determine whether the work falls within the range specified in the FROM *from-value* TO *to-value* clause.

CARDINALITY
The estimated cardinality of the work. This value is used to determine whether the work falls within the range specified in the FROM *from-value* TO *to-value* clause.

FROM *from-value* **TO UNBOUNDED or FROM** *from-value* **TO** *to-value*
Specifies the range of either timeron value (for estimated cost) or cardinality within which the database activity must fall if it is to be part of this work class. The range is inclusive of *from-value* and *to-value*. If this clause is not specified for the work class, all work that falls within the specified work type will be included (that is, the default is FROM 0 TO UNBOUNDED). This range is only used for the following work types:
- ALL
- DML
- READ

- WRITE

FROM *from-value* **TO UNBOUNDED**
The *from-value* must be zero or a positive DOUBLE value (SQLSTATE 5U019). The range has no upper bound.

FROM *from-value* **TO** *to-value*
The *from-value* must be zero or a positive DOUBLE value and the *to-value* must be a positive DOUBLE value. The *from-value* must be smaller than or equal to the *to-value* (SQLSTATE 5U019).

DATA TAG LIST CONTAINS *integer-constant*
Specifies the value of the tag given to any data which the database activity might touch if it is to be part of this work class. If the clause is not specified for the work class, all work that falls within the specified work type, regardless of what data it might touch, will be included (that is, the default is to ignore the data tag). This clause is used only if the work type is READ, WRITE, DML, or ALL and the database activity is a DML statement. Valid values for *integer-constant* are integers from 1 to 9.

schema-clause

ROUTINES IN SCHEMA *schema-name*
Specifies the schema name of the procedure that the CALL statement will be calling. This clause is only used if the work type is CALL or ALL and the database activity is a CALL statement. If no value is specified, all schemas are included.

position-clause

POSITION
Specifies where this work class is to be placed within the work class set, which determines the order in which work classes are evaluated. When performing work class assignment at run time, the database manager first determines the work class set that is associated with the object, either the database or a service superclass. The first matching work class within that work class set is then selected. If this keyword is not specified, the work class is placed in the last position.

LAST
Specifies that the work class is to be placed last in the ordered list of work classes within the work class set. This is the default.

BEFORE *work-class-name*
Specifies that the work class is to be placed before work class *work-class-name* in the list. The *work-class-name* must identify a work class in the work class set that exists at the current server (SQLSTATE 42704).

AFTER *work-class-name*
Specifies that the work class is to be placed after work class *work-class-name* in the list. The *work-class-name* must identify a work class in the work class set that exists at the current server (SQLSTATE 42704).

AT *position*
Specifies the absolute position at which the work class is to be placed within the work class set in the ordered list of work classes. This value can be any positive integer (not zero) (SQLSTATE

42615). If *position* is greater than the number of existing work classes plus one, the work class is placed at the last position within the work class set.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (WORK ACTION SET)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (WORK CLASS SET)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.

Examples

- *Example 1:* Create a work class set named LARGE_QUERIES that has a set of work classes representing all DML with an estimated cost greater than 9999 and an estimated cardinality greater than 1000.

```
CREATE WORK CLASS SET LARGE_QUERIES
  (WORK CLASS LARGE_ESTIMATED_COST WORK TYPE DML
   FOR TIMERONCOST FROM 9999 TO UNBOUNDED,
   WORK CLASS LARGE_CARDINALITY WORK TYPE DML
   FOR CARDINALITY FROM 1000 TO UNBOUNDED)
```

- *Example 2:* Create a work class set named DML_SELECTS that has a work class representing all DML SELECT statements that do not contain a DELETE, INSERT, MERGE, or UPDATE statement.

```
CREATE WORK CLASS SET DML_SELECTS
  (WORK CLASS SELECT_CLASS WORK TYPE READ)
```

CREATE WORKLOAD

The CREATE WORKLOAD statement defines a workload.

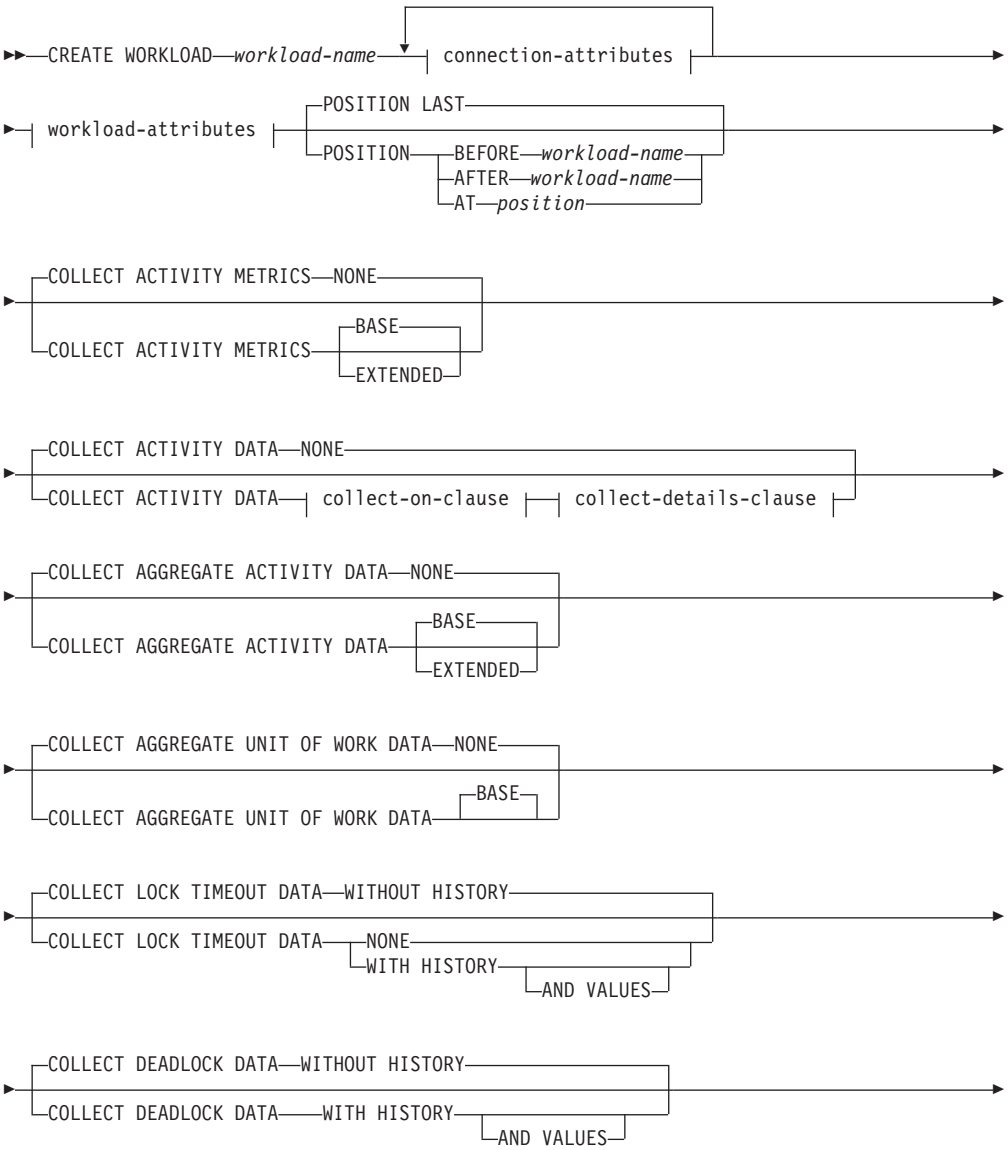
Invocation

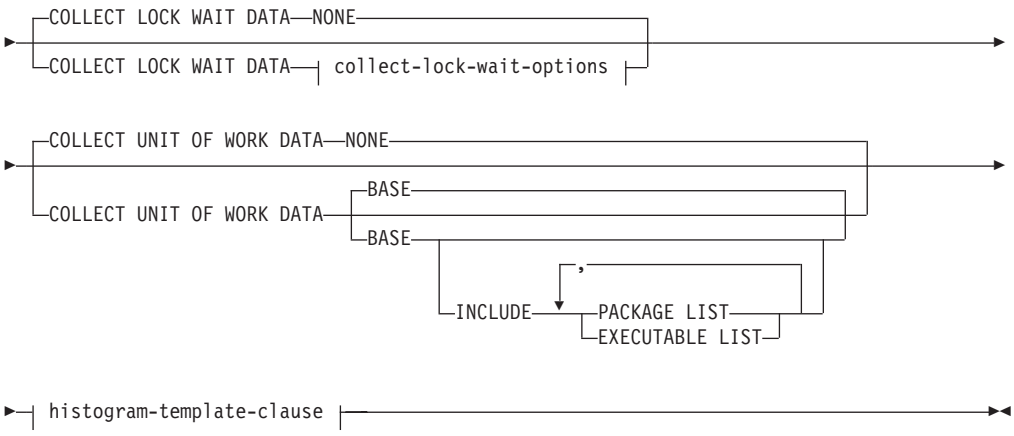
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

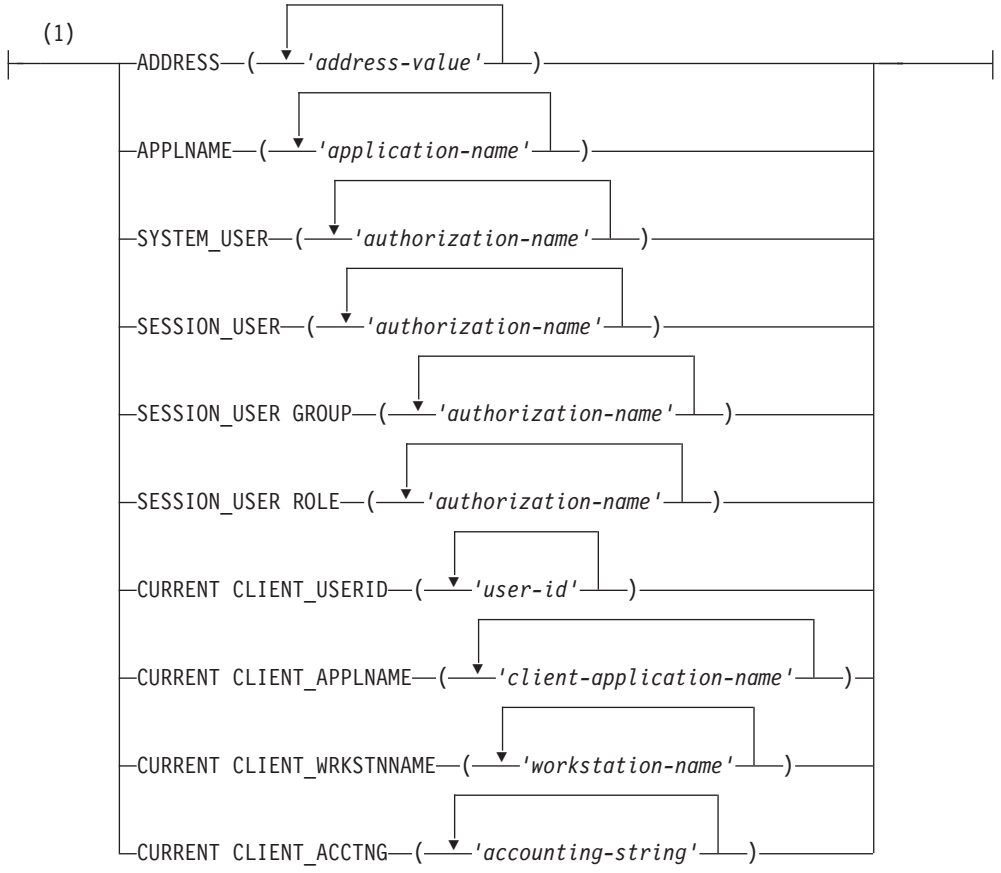
The privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

Syntax

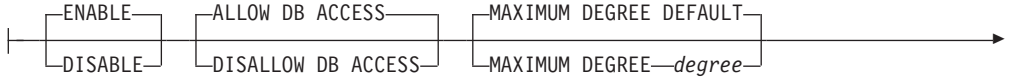


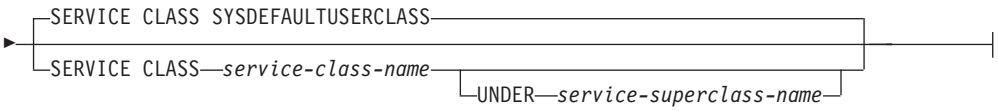


connection-attributes:



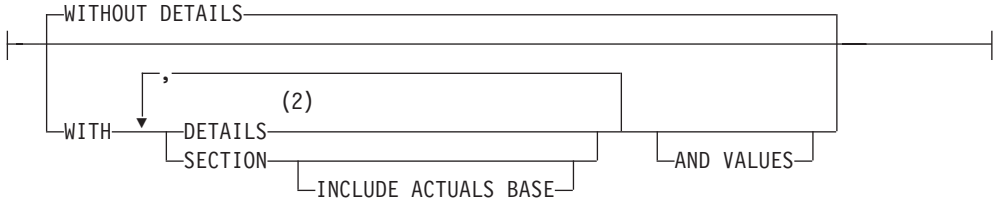
workload-attributes:



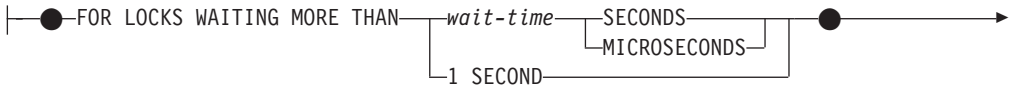


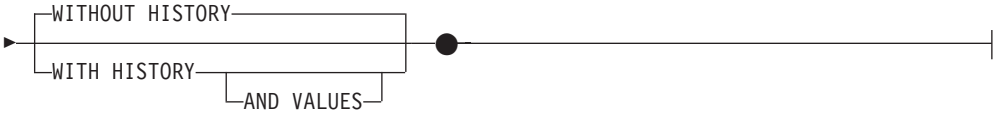
collect-on-clause:

collect-details-clause:

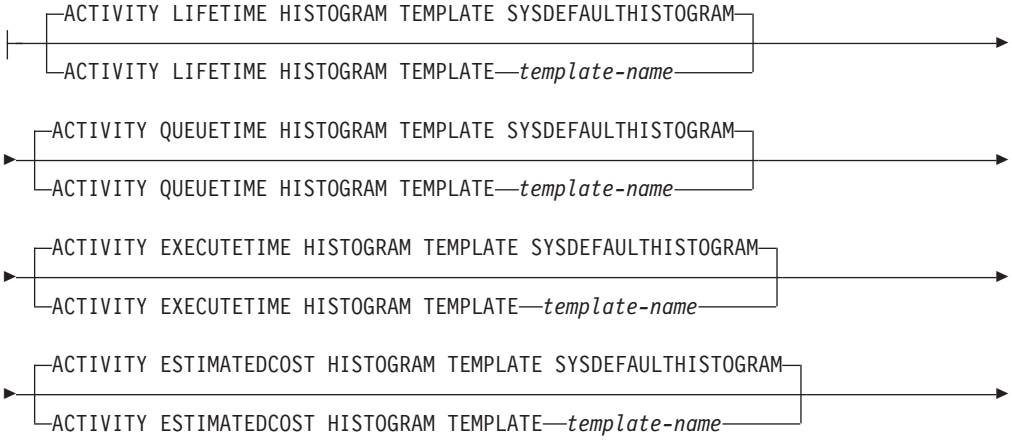


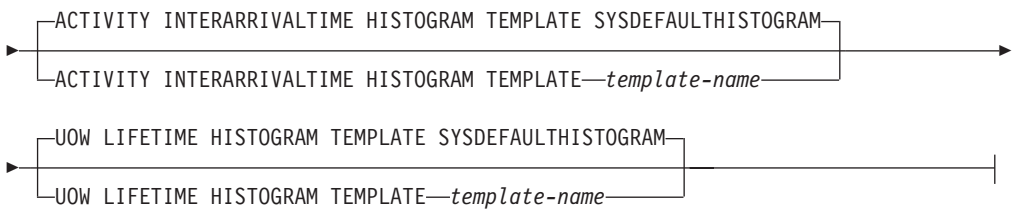
collect-lock-wait-options:





histogram-template-clause:





Notes:

1 Each connection attribute clause can only be specified once.

2 The DETAILS keyword is the minimum to be specified, followed by the option separated by a comma.

Description

workload-name
Names the workload. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *workload-name* must not identify a workload that already exists at the current server (SQLSTATE 42710). The name must not begin with the characters 'SYS' (SQLSTATE 42939).

connection-attributes
The attributes of the connection must match all attributes specified in this workload definition if it is to be associated with this workload when the connection is established. If a list of values is specified for a connection attribute in the workload definition, the corresponding attribute of the connection must match at least one of the values in the list. If a connection attribute is not specified in the workload definition, the connection can have any value for the corresponding connection attribute.

Note: All connection attributes are case sensitive, except for ADDRESS.

ADDRESS ('*address-value***', ...)**
Specifies one or more IPv4 addresses, IPv6 addresses or secure domain names for the ADDRESS connection attribute. An address value cannot appear more than once in the list (SQLSTATE 42713). The address-value must be an IPv4 address, an IPv6 address, or a secure domain name.

An IPv4 address must not contain leading spaces and is represented as a dotted decimal address. An example of an IPv4 address is `192.0.2.1`. The value `localhost` or its equivalent representation `127.0.0.1` will not result in a match; the real IPv4 address of the host must be specified instead. An IPv6 address must not contain leading spaces and is represented as a colon hexadecimal address. An example of an IPv6 address is `2001:0DB8:0000:0000:0008:0800:200C:417A`. IPv4-mapped IPv6 addresses (`::ffff:192.0.2.1`, for example) will not result in a match. Similarly, `localhost` or its IPv6 short representation `::1` will not result in a match. A domain name is converted to an IP address by the domain name server where a resulting IPv4 or IPv6 address is determined. An example of a domain name is `corona.example.com`. When a domain name is converted to an IP address, the result of this conversion could be a set of one or more IP addresses. In this case, an incoming connection is said to match the ADDRESS attribute of a workload object if the IP address from which the connection originates matches any of the IP addresses to which the domain name was converted.

When creating a workload object, you should specify domain name values for the ADDRESS attribute instead of static IP addresses, particularly in Dynamic Host Configuration Protocol (DHCP) environments where a device can have a different IP address each time it connects to the network.

APPLNAME ('*application-name***', ...)**
Specifies one or more applications for the APPLNAME connection attribute. An application name cannot appear more than once in the list (SQLSTATE 42713). If *application-name* does not contain a single asterisk character (*), is equivalent to the value shown in the "Application name" field in system monitor output and in output from the LIST APPLICATIONS command. If *application-name* does contain a single asterisk character (*), the value is used as an expression to represent a set of application names, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the application name, use a sequence of two asterisk characters (**).

SYSTEM_USER ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SYSTEM USER connection attribute. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

SESSION_USER ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SESSION USER connection attribute. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

SESSION_USER GROUP ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SESSION_USER GROUP connection attribute. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

SESSION_USER ROLE ('*authorization-name***', ...)**
Specifies one or more authorization IDs for the SESSION_USER ROLE connection attribute. The roles of a session authorization ID in this context refer to all the roles that are available to the session authorization ID, regardless of how the roles were obtained. An authorization ID cannot appear more than once in the list (SQLSTATE 42713).

CURRENT CLIENT_USERID ('*user-id***', ...)**
Specifies one or more client user IDs for the CURRENT CLIENT_USERID connection attribute. A client user ID cannot appear more than once in the list (SQLSTATE 42713). If *user-id* contains a single asterisk character (*), the value is used as an expression to represent a set of user IDs, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the user ID, use a sequence of two asterisk characters (**).

CURRENT CLIENT_APPLNAME ('*client-application-name***', ...)**
Specifies one or more applications for the CURRENT CLIENT_APPLNAME connection attribute. An application name cannot appear more than once in the list (SQLSTATE 42713). If *client-application-name* does not contain a single asterisk character (*), is equivalent to the value shown in the "TP Monitor client application name" field in system monitor output. If *client-application-name* does contain a single asterisk character (*), the value is used as an expression to represent a set of application names, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the application name, use a sequence of two asterisk characters (**).

CURRENT CLIENT_WRKSTNNAME ('*workstation-name*', ...)
Specifies one or more client workstation names for the CURRENT CLIENT_WRKSTNNAME connection attribute. A client workstation name cannot appear more than once in the list (SQLSTATE 42713). If *workstation-name* contains a single asterisk character (*), the value is used as an expression to represent a set of workstation names, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the workstation name, use a sequence of two asterisk characters (**).

CURRENT CLIENT_ACCTNG ('*accounting-string*', ...)
Specifies one or more client accounting strings for the CURRENT CLIENT_ACCTNG connection attribute. A client accounting string cannot appear more than once in the list (SQLSTATE 42713). If *accounting-string* contains a single asterisk character (*), the value is used as an expression to represent a set of accounting strings, where the asterisk (*) represents a string of zero or more characters. If the expression needs to include an asterisk character in the accounting string, use a sequence of two asterisk characters (**).

workload-attributes
Specifies attributes of the workload.

ENABLE or DISABLE
Specifies whether or not this workload will be considered when a workload is chosen. The default is ENABLE.

ENABLE
Specifies that the workload is enabled and will be considered when a workload is chosen.

DISABLE
Specifies that the workload is disabled and will not be considered when a workload is chosen.

ALLOW DB ACCESS or DISALLOW DB ACCESS
Specifies whether or not a workload occurrence associated with this workload is allowed access to the database. The default is ALLOW DB ACCESS.

ALLOW DB ACCESS
Specifies that workload occurrences associated with this workload are allowed access to the database.

DISALLOW DB ACCESS
Specifies that workload occurrences associated with this workload are not allowed access to the database. The next unit of work associated with this workload will be rejected (SQLSTATE 5U020). Workload occurrences that are already running are allowed to complete.

MAXIMUM DEGREE
Specifies the maximum runtime degree of parallelism for this workload. The default is DEFAULT.

DEFAULT
Specifies that this workload inherits the intrapartition parallelism setting from the database manager configuration parameter **intra_parallel**. When **intra_parallel** is set to NO, this workload runs with intrapartition parallelism disabled. When **intra_parallel** is set to YES, this workload runs with intrapartition parallelism enabled. This workload does not specify a maximum runtime degree for assigned

applications. Therefore, the actual runtime degree is determined as the lower of the value of **max_querydegree** configuration parameter, the value set by SET RUNTIME DEGREE command, and the SQL statement compilation degree.

degree
Specifies the maximum degree of parallelism for this workload. Valid values are 1 to 32,767. With value 1, the associated requests run with intrapartition parallelism disabled. With value 2 to 32,767, the associated requests run with intrapartition parallelism enabled. The actual runtime degree is determined as the lower of this *degree*, the value of **max_querydegree** configuration parameter, the value set by SET RUNTIME DEGREE command and the SQL statement compilation degree.

Note: A MAXIMUM DEGREE value greater than 1 will not enable intrapartition parallelism unless the shared sort heap is available.

SERVICE CLASS *service-class-name*
Specifies that requests associated with this workload are to be executed in the service class *service-class-name*. The *service-class-name* must identify a service class that exists at the current server (SQLSTATE 42704). The *service-class-name* cannot be 'SYSDEFAULTSUBCLASS', 'SYSDEFAULTSYSTEMCLASS', or 'SYSDEFAULTMAINTENANCECLASS' (SQLSTATE 5U032). The default is SYSDEFAULTUSERCLASS.

UNDER *service-superclass-name*
This clause is used when specifying a service subclass. The *service-superclass-name* identifies the service superclass of *service-class-name*. The *service-superclass-name* must identify a service superclass that exists at the current server (SQLSTATE 42704). The *service-superclass-name* cannot be 'SYSDEFAULTSYSTEMCLASS' or 'SYSDEFAULTMAINTENANCECLASS' (SQLSTATE 5U032).

POSITION
Specifies where this workload is to be placed within the ordered list of workloads. At run time, this list is searched in order for the first workload that matches the required connection attributes. The default is LAST.

LAST
Specifies that the workload is to be last in the list, before the default workloads SYSDEFAULTUSERWORKLOAD and SYSDEFAULTADMWORKLOAD.

BEFORE *relative-workload-name*
Specifies that the workload is to be placed before workload *relative-workload-name* in the list. The *relative-workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The BEFORE option cannot be specified if *relative-workload-name* is 'SYSDEFAULTUSERWORKLOAD' or 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

AFTER *relative-workload-name*
Specifies that the workload is to be placed after workload *relative-workload-name* in the list. The *relative-workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The AFTER option cannot be specified if *relative-workload-name* is 'SYSDEFAULTUSERWORKLOAD' or 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

AT *position*
Specifies the absolute position at which the workload is to be placed in the list. This value can be any positive integer (not zero) (SQLSTATE 42615). If *position* is greater than the number of existing workloads plus one, the workload is placed at the last position, just before SYSDEFAULTUSERWORKLOAD and SYSDEFAULTADMWORKLOAD.

COLLECT ACTIVITY METRICS
Specifies that monitor metrics should be collected for an activity submitted by an occurrence of the workload. The default is COLLECT ACTIVITY METRICS NONE.

Note: The effective activity metrics collection setting is the combination of the attribute specified by the COLLECT ACTIVITY METRICS clause on the workload submitting the activity, and the **mon_act_metrics** database configuration parameter. If either the workload attribute or the configuration parameter has a value other than NONE, metrics will be collected for the activity.

NONE
Specifies that no metrics will be collected for any activity submitted by an occurrence of the workload.

BASE
Specifies that basic metrics will be collected for any activity submitted by an occurrence of the workload.

EXTENDED
Specifies that basic metrics will be collected for any activity submitted by an occurrence of the workload. In addition, specifies that the values for the following monitor elements should be determined with additional granularity:

- **total_section_time**
- **total_section_proc_time**
- **total_routine_user_code_time**
- **total_routine_user_code_proc_time**
- **total_routine_time**

COLLECT ACTIVITY DATA
Specifies that data about each activity associated with this workload is to be sent to any active activities event monitor when the activity completes. The default is COLLECT ACTIVITY DATA NONE.

collect-on-clause
Specifies where the activity data is to be collected. The default is ON COORDINATOR MEMBER.

ON COORDINATOR MEMBER
Specifies that activity data is to be collected only at the coordinator member of the activity.

ON ALL MEMBERS
Specifies that activity data is to be collected at all members where the activity is processed. On remote members, a record for the activity may be captured multiple times as the activity comes and goes on those members. If the AND VALUES clause is specified, activity input values will be collected only for the members of the coordinator.

NONE
Specifies that activity data is not collected for each activity that is associated with this workload.

collect-details-clause
Specifies what type of activity data is to be collected. The default is WITHOUT DETAILS.

WITHOUT DETAILS
Specifies that data about each activity that is associated with this workload is to be sent to any active activities event monitor, when the activity completes execution. Details about statement, compilation environment, and section environment data are not sent.

WITH

DETAILS
Specifies that statement and compilation environment data is to be sent to any active activities event monitor, for those activities that have them. Section environment data is not sent.

SECTION
Specifies that statement, compilation environment, section environment data, and section actuals are to be sent to any active activities event monitor for those activities that have them. DETAILS must be specified if SECTION is specified. Section actuals will be collected on any member where the activity data is collected.

INCLUDE ACTUALS BASE
Specifies that section actuals should also be collected on any partition where the activity data is collected. For section actuals to be collected, either INCLUDE ACTUALS clause must be specified or the **section_actuals** database configuration parameter must be set.

The effective setting for the collection of section actuals is the combination of the INCLUDE ACTUALS clause, the **section_actuals** database configuration parameter, and the <collectsectionactuals> setting specified on the WLM_SET_CONN_ENV routine. For example, if INCLUDE ACTUALS BASE is specified, yet the **section_actuals** database configuration parameter value is NONE and <collectsectionactuals> is set to NONE, then the effective setting for the collection of section actuals is BASE.

BASE specifies that the following should be enabled and collected during the activity's execution:

- Basic operator cardinality counts
- Statistics for each object referenced (DML statements only)

AND VALUES
Specifies that input data values are to be sent to any active activities event monitor, for those activities that have them.

COLLECT AGGREGATE ACTIVITY DATA
Specifies that aggregate activity data about the activities associated with this workload is to be sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default when

COLLECT AGGREGATE ACTIVITY DATA is not specified is COLLECT AGGREGATE ACTIVITY DATA NONE. The default when COLLECT AGGREGATE ACTIVITY DATA is specified is COLLECT AGGREGATE ACTIVITY DATA BASE.

BASE
Specifies that basic aggregate activity data about the activities associated with this workload is to be sent to the statistics event monitor, if one is active. Basic aggregate activity data includes:

- Activity CPU time high watermark
- Activity execution time histogram
- Activity life time histogram
- Activity queue time histogram
- Activity rows read high watermark
- Estimated activity cost high watermark
- Rows returned high watermark
- Temporary table space usage high watermark. Only activities that have an SQLTEMPSPACE threshold applied to them participate in this high watermark.

EXTENDED
Specifies that all aggregate activity data about the activities associated with this workload is to be sent to the statistics event monitor, if one is active. This includes all basic aggregate activity data plus:

- Activity data manipulation language (DML) estimated cost histogram
- Activity DML inter-arrival time histogram

NONE
Specifies that no aggregate activity data is to be collected for this workload.

COLLECT AGGREGATE UNIT OF WORK DATA
Specifies that aggregate unit of work data about the units of work associated with this workload is to be sent to the statistics event monitor, if one is active. This information is collected periodically on an interval that is specified by the **wlm_collect_int** database configuration parameter. The default when COLLECT AGGREGATE UNIT OF WORK DATA is not specified is COLLECT AGGREGATE UNIT OF WORK DATA NONE.

BASE
Specifies that basic aggregate unit of work data about the units of work associated with this workload is to be sent to the statistics event monitor, if one is active. Basic aggregate unit of work includes:

- Unit of work lifetime histogram

NONE
Specifies that no aggregate unit of work data is to be collected for this workload.

COLLECT LOCK TIMEOUT DATA
Specifies that data about lock timeout events that occur within this workload is sent to the applicable event monitor when the lock event occurs. The lock timeout data is collected on all members. The default is COLLECT LOCK TIMEOUT DATA WITHOUT HISTORY. This setting works in conjunction with the **mon_locktimeout** database configuration parameter setting. The setting that produces the most detailed output is honored.

WITHOUT HISTORY
Specifies that data about lock events that occur within this workload is sent to any active locking event monitor when the lock event occurs. Past activity history and input values are not sent to the event monitor.

NONE
Specifies that lock timeout data for the workload is not collected at any member.

WITH HISTORY
Specifies to collect past activity history in the current unit of work for all of this type of lock events. The activity history buffer will wrap after the maximum size limit is used.

The default limit on the number of past activities to be kept by any one application is 250. If the number of past activities is greater than the limit, only the newest activities are reported. This default value can be overridden using the registry variable DB2_MAX_INACT_STMTS to specify a different value. You can choose a different value for the limit to increase or reduce the amount of system monitor heap used for past activity information.

AND VALUES
Specifies that input data values are to be sent to any active locking event monitor for those activities that have them. These data values will not include LOB data, LONG VARCHAR data, LONG VARGRAPHIC data, structured type data, or XML data. For SQL statements compiled using the REOPT ALWAYS bind option, there will be no REOPT compilation or statement execution data values provided in the event information.

COLLECT DEADLOCK DATA
Specifies that data about deadlock events that occur within this workload is sent to any active locking event monitor when the lock event occurs. The deadlock data is collected on all members. The default is COLLECT DEADLOCK DATA WITHOUT HISTORY. This setting is only honored if the **mon_deadlock** database configuration parameter is not set to NONE.

WITHOUT HISTORY
Specifies that data about lock events that occur within this workload is sent to any active locking event monitor when the lock event occurs. Past activity history and input values are not sent to the event monitor.

WITH HISTORY
Specifies to collect past activity history in the current unit of work for all of this type of lock events. The activity history buffer will wrap after the maximum size limit is used.

The default limit on the number of past activities to be kept by any one application is 250. If the number of past activities is greater than the limit, only the newest activities are reported. This default value can be overridden using the registry variable DB2_MAX_INACT_STMTS to specify a different value. You can choose a different value for the limit to increase or reduce the amount of system monitor heap used for past activity information.

AND VALUES
Specifies that input data values are to be sent to any active locking event monitor for those activities that have them. These data values will not include LOB data, LONG VARCHAR data, LONG

VARGRAPHIC data, structured type data, or XML data. For SQL statements compiled using the REOPT ALWAYS bind option, there will be no REOPT compilation or statement execution data values provided in the event information.

COLLECT LOCK WAIT DATA
Specifies that data about lock wait events that occur within this workload is sent to any active locking event monitor when the lock has not been acquired within *wait-time*. The default is COLLECT LOCK WAIT DATA NONE with a default *wait-time* value of 0 microseconds. This setting works in conjunction with the **mon_lockwait** and **mon_lw_thresh** database configuration parameters. The setting that produces the most detailed output is honored.

NONE
Specifies that the lock wait event for the workload is not collected at any member.

FOR LOCKS WAITING MORE THAN *wait-time* **(SECONDS | MICROSECONDS) | 1 SECOND**
Specifies that data about lock wait events that occur within this workload is sent to any active locking event monitor when the lock has not been acquired within *wait-time*.

This value can be any non-negative integer. Use a valid duration keyword to specify an appropriate unit of time for *wait-time*. The minimum valid value for the *wait-time* parameter is 1000 microseconds.

WITH HISTORY
Specifies to collect past activity history in the current unit of work for all of this type of lock events. The activity history buffer will wrap after the maximum size limit is used.

The default limit on the number of past activities to be kept by any one application is 250. If the number of past activities is greater than the limit, only the newest activities are reported. This default value can be overridden using the registry variable DB2_MAX_INACT_STMTS to specify a different value. You can choose a different value for the limit to increase or reduce the amount of system monitor heap used for past activity information.

AND VALUES
Specifies that input data values are to be sent to any active locking event monitor for those activities that have them. These data values will not include LOB data, LONG VARCHAR data, LONG VARGRAPHIC data, structured type data, or XML data. For SQL statements compiled using the REOPT ALWAYS bind option, there will be no REOPT compilation or statement execution data values provided in the event information.

COLLECT UNIT OF WORK DATA
Specifies that data about each transaction associated with this workload is to be sent to the unit of work event monitor, if any are active, when the unit of work ends. The default, when COLLECT UNIT OF WORK DATA is not specified, is COLLECT UNIT OF WORK DATA NONE. The default, when COLLECT UNIT OF WORK DATA is specified, is COLLECT UNIT OF WORK DATA BASE. If the **mon_uow_data** database configuration parameter is set to BASE, it takes precedence over the COLLECT UNIT OF WORK DATA parameter. A value of NONE for the **mon_uow_data** indicates that the COLLECT UNIT OF WORK DATA parameters of individual workloads is used.

NONE
Specifies that no unit of work data for transactions associated with this workload is sent to the unit of work event monitor. The default is COLLECT UNIT OF WORK DATA NONE.

BASE
Specifies that base level of data for transactions associated with this workload is sent to the unit of work event monitors.

Some of the information reported in a unit of work event are system level request metrics. The collection of these metrics is controlled independently from the collection of the unit of work data. The request metrics are controlled with the COLLECT REQUEST METRICS clause on superclass, or using the **mon_req_metrics** database configuration parameter. The service super class which the workload is associated with, or the service super class of the service subclass which the workload is associated with, must have the collection of request metrics enabled in order for the request metrics to be present in the unit of work event. If the request metrics collection is not enabled, the value of the request metrics will be zero.

INCLUDE PACKAGE LIST
Specifies that base level of data and the package list for transactions associated with this workload are sent to the unit of work event monitor.

The size of the collected package list is determined by the value of the **mon_pkglist_sz** database configuration parameter. If this value is 0, then the package list is not collected even if the `PACKAGE LIST` option is specified.

In a partitioned database environment, the package list is only available on the coordinator member. The BASE level will be collected on remote members.

Some of the information reported in a unit of work event are system level request metrics. The collection of these metrics is controlled independently from the collection of the unit of work data. The request metrics are controlled with the COLLECT REQUEST METRICS clause on superclass, or using the **mon_req_metrics** database configuration parameter. The service super class which the workload is associated with, or the service super class of the service subclass which the workload is associated with, must have the collection of request metrics enabled in order for the request metrics to be present in the unit of work event. If the request metrics collection is not enabled, the value of the request metrics will be zero.

INCLUDE EXECUTABLE LIST
Specifies that executable ID list will be collected for a unit of work together with base level of data and sent to the unit of work event monitor.

histogram-template-clause
Specifies the histogram templates to use when collecting aggregate activity data for activities executing in the workload.

ACTIVITY LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of DB2 activities running in the workload during a specific interval. This time includes both time queued and time executing. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY QUEUETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the workload are queued during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option.

ACTIVITY EXECUTETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, that DB2 activities running in the workload are executing during a specific interval. This time does not include the time spent queued. Activity execution time is collected in this histogram at the coordinator member only. The time does not include idle time. Idle time is the time between the execution of requests belonging to the same activity when no work is being done. An example of idle time is the time between the end of opening a cursor and the start of fetching from that cursor. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified, with either the BASE or EXTENDED option. Only activities at nesting level 0 are considered for inclusion in the histogram.

ACTIVITY ESTIMATEDCOST HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the estimated cost, in timerons, of DML activities running in the workload. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option. Only activities at nesting level 0 are considered for inclusion in the histogram.

ACTIVITY INTERARRIVALTIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the length of time, in milliseconds, between the arrival of one DML activity into this workload and the arrival of the next DML activity into this workload. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE ACTIVITY DATA clause is specified with the EXTENDED option.

UOW LIFETIME HISTOGRAM TEMPLATE *template-name*
Specifies the template that describes the histogram used to collect statistical data about the duration, in milliseconds, of units of work running in the workload during a specific interval. The default is SYSDEFAULTHISTOGRAM. This information is collected only when the COLLECT AGGREGATE UNIT OF WORK DATA clause is specified with the BASE option.

Rules

- A workload management (WLM)-exclusive SQL statement must be followed by a COMMIT or a ROLLBACK statement (SQLSTATE 5U021). WLM-exclusive SQL statements are:
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)

  - CREATE WORK ACTION SET, ALTER WORK ACTION SET, or DROP (WORK ACTION SET)
  - CREATE WORK CLASS SET, ALTER WORK CLASS SET, or DROP (WORK CLASS SET)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- A WLM-exclusive SQL statement cannot be issued within a global transaction (SQLSTATE 51041) such as, for example, an XA transaction.

Notes

- Changes are written to the system catalog, but do not take effect until they are committed, even for the connection that issues the statement.
- Only one uncommitted WLM-exclusive SQL statement at a time is allowed across all partitions. If an uncommitted WLM-exclusive SQL statement is executing, subsequent WLM-exclusive SQL statements will wait until the current WLM-exclusive SQL statement commits or rolls back.
- When a database connection is established, the database manager looks for a matching workload based on the connection attributes that were specified in the POSITION clause (in order of specification). If a matching workload is found, the database manager checks whether the current session user has USAGE privilege on that workload. If the session user does not have USAGE privilege on the workload, the database manager looks for the next matching workload. If the session user has USAGE privilege on this workload, the connection is associated with the workload. If a matching workload is not found, the connection is associated with the default user workload, SYSDEFAULTUSERWORKLOAD. If the session user does not have USAGE privilege on SYSDEFAULTUSERWORKLOAD, an error is returned (SQLSTATE 42501).
- The workload association is re-evaluated at the beginning of each new unit of work if the database manager detects one of the following conditions.
  - The connection attributes have changed. This can happen if any of the following events has occurred:
    - The set client information API (sqleseti) has been invoked and it changed the connection attributes that were included in the workload definition. Note that although the client information can be set by the end user so that it could initiate a workload re-evaluation, the workload remapping itself cannot happen if the session user does not have the USAGE privilege on the workload.
    - The SET SESSION AUTHORIZATION statement has been invoked and it changed the current session user.
    - The roles that are available to a session user have changed.
  - A workload is created.
  - A workload is dropped.
  - A workload is altered.
  - The USAGE privilege on a workload is granted to a user, group, or role.
  - The USAGE privilege on a workload is revoked from a user, group, or role.

  If the workload re-evaluation results in no workload reassignment, the current workload occurrence continues to run; that is, a new workload occurrence will not be started.
- A connection cannot be reassigned to a different workload when an activity is still active. Examples of such activities are a load operation, an executing

procedure, or statements that maintain resources across multiple units of work, such as an open WITH HOLD cursor. The current workload occurrence continues to run until all executing activities complete. Workload reassignment occurs at the beginning of the next unit of work.

- After a service class has been referenced by a workload, it cannot be dropped until it is no longer referenced by any workload. Either of the following actions can be taken to remove a service class reference from a workload:
  - Alter the workload to change the service class name
  - Drop the workload
- After a role has been referenced by a workload, it cannot be dropped until it is no longer referenced by any workload. Either of the following actions can be taken to remove a role reference from a workload:
  - Alter the workload to remove the role
  - Drop the workload
- ***Privileges***: The USAGE privilege is not granted to any user, group, or role when a workload is created. To enable use of a workload, grant USAGE privilege on that workload to a user, a group, or a role using the GRANT USAGE ON WORKLOAD statement.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DATABASE PARTITION can be specified in place of MEMBER, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - DATABASE PARTITIONS can be specified in place of MEMBERS, except when the DB2_ENFORCE_MEMBER_SYNTAX registry variable is set to ON.
  - COLLECT UNIT OF WORK DATA PACKAGE LIST can be specified in place of COLLECT UNIT OF WORK DATA BASE INCLUDE PACKAGE LIST.

Examples

- *Example 1:* Create a workload named CAMPAIGN for requests that are submitted by a session user belonging to group FINANCE. These requests are to be executed in the default user service class SYSDEFAULTUSERCLASS.

```
CREATE WORKLOAD CAMPAIGN
  SESSION_USER GROUP ('FINANCE')
```

- *Example 2:* Create a workload named PAYROLL for a session user with role HR that has the CURRENT CLIENT_APPLNAME special register set to SALARYSYS. Units of work associated with this workload are to be executed in service class MEDIUMSC that is under the service superclass HRSC. When a workload is chosen at run time, this workload should be evaluated only after the workload CAMPAIGN has been evaluated and determined to not match.

```
CREATE WORKLOAD PAYROLL
  SESSION_USER ROLE ('HR')
  CURRENT CLIENT_APPLNAME ('SALARYSYS') SERVICE CLASS MEDIUMSC
  UNDER HRSC POSITION AFTER CAMPAIGN
```

- *Example 3:* An occurrence of workload CAMPAIGN (from example 1) is currently running on the system. Create a workload named NEWCAMPAIGN, also for requests that are submitted by a session user belonging to group FINANCE, but only those requests submitted through application DB2BP.EXE. Requests associated with this workload are to be executed in service class MARKETINGSC. NEWCAMPAIGN should be evaluated before CAMPAIGN.

```
CREATE WORKLOAD NEWCAMPAIGN
  SESSION_USER GROUP ('FINANCE')
  APPLNAME ('DB2BP.EXE') SERVICE CLASS MARKETINGSC
  POSITION BEFORE CAMPAIGN
```

The running workload occurrence of CAMPAIGN continues to run until the current unit of work completes, at which time a workload re-evaluation takes place, and the connection could then be remapped to workload NEWCAMPAIGN.

- *Example 4:* Create a workload named REPORTS for requests that are submitted through application appl1, appl2, or appl3 by system user BOB or MARY.

```
CREATE WORKLOAD REPORTS
  APPLNAME ('appl1', 'appl2', 'appl3')
  SYSTEM_USER ('BOB', 'MARY')
```

- *Example 5:* Assuming a lock event monitor called PAYROLL exists and is active, create lock event records with statement history for lock timeout events that occur within the workload EMPLOYEES.

```
CREATE WORKLOAD EMPLOYEES
  APPLNAME ("app1", "app2")
  COLLECT LOCK TIMEOUT DATA WITH HISTORY
```

- *Example 6:* Assuming a lock event monitor called PAYROLL exists and is active, create lock event records for only deadlock and lock timeout events that occur within the workload FINANCE on all partitions.

```
CREATE WORKLOAD FINANCE
  APPLNAME ("app1", "app2")
  COLLECT DEADLOCK DATA
  COLLECT LOCK TIMEOUT DATA
```

- *Example 7:* Assuming a lock event monitor called PAYROLL exists and is active, create lock event records with statement history and values for deadlock events that occur within the workload MANAGERS.

```
CREATE WORKLOAD MANAGERS
  APPLNAME ("app1", "app2")
  COLLECT DEADLOCK DATA WITH HISTORY AND VALUES
```

- *Example 8:* Assuming a lock event monitor called PAYROLL exists and is active, create lock event records with statement history for locks that are acquired after waiting 5000 milliseconds within the MANAGERS workload.

```
CREATE WORKLOAD MANAGERS
  APPLNAME ("app1", "app2")
  COLLECT LOCK WAIT DATA FOR LOCKS WAITING MORE THAN 5 SECONDS WITH HISTORY
```

- *Example 9*: Create a workload named ACCRECS for all accounts receivable applications that share a similar name (*accrec01, accrec02 ... accrec15*) and assign them to the service class ACCOUNTNGSC. Application names are identified through the APPLNAME connection attribute with the help of a wild card (*) and do not need to be specified individually.

```
CREATE WORKLOAD ACCRECS
  SESSION_USER GROUP ('ACCOUNTING')
  APPLNAME ('accrec*')
  SERVICE CLASS ACCOUNTNGSC
```

- *Example 10:* Create a workload named CAMPAIGN for requests submitted through the application appl1, and have unit of work data collected and sent to any active unit of work event monitors.

```
CREATE WORKLOAD CAMPAIGN
  APPLNAME ('appl1')
  COLLECT UNIT OF WORK DATA BASE
```

- *Example 11:* The following statements show how you can specify the different address value formats supported by the ADDRESS connection attribute when creating a workload.
  - To specify a secure domain name:

    ```
    CREATE WORKLOAD DOMAINWORKLOAD
      ADDRESS ('aviator.example.com')
    ```

  - To specify a IPv4 address value:

    ```
    CREATE WORKLOAD IPWORKLOAD1
      ADDRESS ('192.0.2.11')
    ```

  - To specify a IPv6 address value (long format):

    ```
    CREATE WORKLOAD IPWORKLOAD2
      ADDRESS ('2001:db8:519:13:204:acff:fe57:6135')
    ```

  - To specify a IPv6 address value (short format):

    ```
    CREATE WORKLOAD IPWORKLOAD3
      ADDRESS ('2001:db8::202:55ff:fe9a:6eee')
    ```

CREATE WRAPPER

The CREATE WRAPPER statement registers a wrapper with a federated server. A wrapper is a mechanism by which a federated server can interact with certain types of data sources.

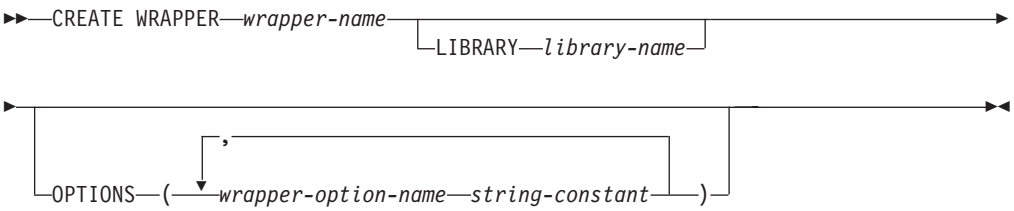
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include DBADM authority.

Syntax



Description

wrapper-name
Names the wrapper. It can be:

- A predefined name. If a predefined name is specified, the federated server automatically assigns a default value to *library-name*.
- A user-supplied name. If a user-supplied name is provided, it is necessary to also specify the appropriate *library-name* to be used with that wrapper and operating system.

LIBRARY *library-name*
Names the file that contains the wrapper library module.

The library name can be specified as an absolute path name or simply the base name (without the path). If only the base name is specified, the library should reside in the `lib` (UNIX) or the `bin` (Windows) subdirectory of the DB2 install path. The *library-name* must be enclosed in single quotation marks.

The LIBRARY option is only necessary when a user-supplied *wrapper-name* is used. This option should not be used when a predefined *wrapper-name* is given.

OPTIONS
Indicates what wrapper options are to be enabled or reset.

wrapper-option-name
Names a wrapper option that is to be enabled or reset.

string-constant
Specifies the setting for *wrapper-option-name* as a character string constant.

The *string-constant* must be enclosed in single quotation marks. Some wrapper options can be used by all wrappers and some options are specific to a particular wrapper.

Notes

- **Syntax alternatives:** The following syntax is supported for compatibility with previous versions of DB2:
  - ADD can be specified before *wrapper-option-name string-constant*.

Examples

- *Example 1*: Register the NET8 wrapper on a federated server to access Oracle data sources. *NET8* is the predefined name for the wrapper that you can use to access Oracle data sources.

```
CREATE WRAPPER NET8
```

- *Example 2*: Register a wrapper on a DB2 federated server that uses the Linux operating system to access ODBC data sources. Assign the name *odbc* to the wrapper that is being registered in the federated database. The full path of the library that contains the ODBC Driver Manager is defined in the wrapper option *MODULE '/usr/lib/odbc.so'*.

```
CREATE WRAPPER odbc OPTIONS (MODULE '/usr/lib/odbc.so')
```

- *Example 3*: Register a wrapper on a DB2 federated server that uses the Windows operating system to access ODBC data sources. The library name for the ODBC wrapper is *'db2rcodbc.dll'*.

```
CREATE WRAPPER odbc LIBRARY 'db2rcodbc.dll'
```

DECLARE CURSOR

The DECLARE CURSOR statement defines a cursor.

Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is not an executable statement and cannot be dynamically prepared. When invoked using the command line processor, additional options can be specified. For more information, refer to “Using command line SQL statements and XQuery statements”.

Authorization

The term “SELECT statement of the cursor” is used to specify the authorization rules. The SELECT statement of the cursor is one of the following statements:

- The prepared select-statement identified by *statement-name*
- The specified *select-statement*

The privileges held by the authorization ID of the statement must include the privileges necessary to execute the *select-statement*. See the Authorization section in "SQL queries".

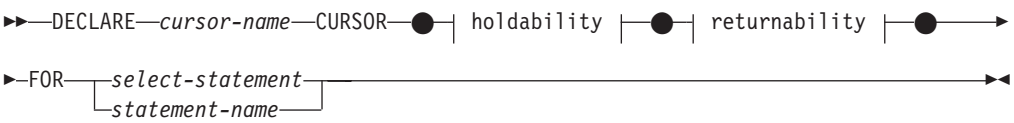
If *statement-name* is specified:

- The authorization ID of the statement is the runtime authorization ID.
- The authorization check is performed when the SELECT-statement is prepared.
- The cursor cannot be opened unless the SELECT-statement is in a prepared state.

If *select-statement* is specified:

- GROUP privileges are not checked.
- The authorization ID of the statement is the authorization ID specified during program preparation.

Syntax



holdability:

returnability:

Description

cursor-name
Specifies the name of the cursor created when the source program is run. The name must not be the same as the name of another cursor declared in the source program. The cursor must be opened before use.

WITHOUT HOLD or WITH HOLD
Specifies whether or not the cursor should be prevented from being closed as a consequence of a commit operation.

WITHOUT HOLD
Does not prevent the cursor from being closed as a consequence of a commit operation. This is the default.

WITH HOLD
Maintains resources across multiple units of work. The effect of the WITH HOLD cursor attribute is as follows:

- For units of work ending with COMMIT:
  - Open cursors defined WITH HOLD remain open. The cursor is positioned before the next logical row of the results table.

    If a DISCONNECT statement is issued after a COMMIT statement for a connection with WITH HOLD cursors, the held cursors must be explicitly closed or the connection will be assumed to have performed work (simply by having open WITH HELD cursors even though no SQL statements were issued) and the DISCONNECT statement will fail.
  - All locks are released, except locks protecting the current cursor position of open WITH HOLD cursors. The locks held include the locks on the table, and for parallel environments, the locks on rows where the cursors are currently positioned. Locks on packages and dynamic SQL sections (if any) are held.
  - Valid operations on cursors defined WITH HOLD immediately following a COMMIT request are:
    - FETCH: Fetches the next row of the cursor.
    - CLOSE: Closes the cursor.
  - UPDATE and DELETE CURRENT OF CURSOR are valid only for rows that are fetched within the same unit of work.
  - LOB locators are freed.
  - The set of rows modified by:
    - A data change statement
    - Routines that modify SQL data embedded within open WITH HOLD cursors

    is committed.
- For units of work ending with ROLLBACK:

- All open cursors are closed.
- All locks acquired during the unit of work are released.
- LOB locators are freed.

• For special COMMIT case:
- Packages can be recreated either explicitly, by binding the package, or implicitly, because the package has been invalidated and then dynamically recreated the first time it is referenced. All held cursors are closed during package rebind. This might result in errors during subsequent execution.

WITHOUT RETURN or WITH RETURN
Specifies whether or not the result table of the cursor is intended to be used as a result set that will be returned from a procedure.

WITHOUT RETURN
Specifies that the result table of the cursor is not intended to be used as a result set that will be returned from a procedure.

WITH RETURN
Specifies that the result table of the cursor is intended to be used as a result set that will be returned from a procedure. WITH RETURN is relevant only if the DECLARE CURSOR statement is contained with the source code for a procedure. In other cases, the precompiler might accept the clause, but it has no effect.

Within an SQL procedure, cursors declared using the WITH RETURN clause that are still open when the SQL procedure ends, define the result sets from the SQL procedure. All other open cursors in an SQL procedure are closed when the SQL procedure ends. Within an external procedure (one not defined using LANGUAGE SQL), the default for all cursors is WITH RETURN TO CALLER. Therefore, all cursors that are open when the procedure ends will be considered result sets. Cursors that are returned from a procedure cannot be declared as scrollable cursors.

TO CALLER
Specifies that the cursor can return a result set to the caller. For example, if the caller is another procedure, the result set is returned to that procedure. If the caller is a client application, the result set is returned to the client application.

TO CLIENT
Specifies that the cursor can return a result set to the client application. This cursor is invisible to any intermediate nested procedures. If a function, method, or trigger called the procedure either directly or indirectly, result sets cannot be returned to the client and the cursor will be closed after the procedure finishes.

select-statement
Identifies the SELECT statement of the cursor. The *select-statement* must not include parameter markers, but can include references to host variables. The declarations of the host variables must precede the DECLARE CURSOR statement in the source program.

statement-name
The SELECT statement of the cursor is the prepared SELECT statement identified by the *statement-name* when the cursor is opened. The *statement-name* must not be identical to a *statement-name* specified in another DECLARE CURSOR statement of the source program.

For an explanation of prepared SELECT statements, see "PREPARE".

Notes

- A program called from another program, or from a different source file within the same program, cannot use the cursor that was opened by the calling program.
- Unnested procedures, with LANGUAGE other than SQL, will have WITH RETURN TO CALLER as the default behavior if DECLARE CURSOR is specified without a WITH RETURN clause, and the cursor is left open in the procedure. This provides compatibility with procedures from previous versions that allow procedures to return result sets to applicable client applications. To avoid this behavior, close all cursors opened in the procedure.
- If the SELECT statement of a cursor contains CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP, all references to these special registers will yield the same respective datetime value on each FETCH. This value is determined when the cursor is opened.
- For more efficient processing of data, the database manager can block data for read-only cursors when retrieving data from a remote server. The use of the FOR UPDATE clause helps the database manager decide whether a cursor is updatable or not. Updatability is also used to determine the access path selection as well. If a cursor is not going to be used in a Positioned UPDATE or DELETE statement, it should be declared as FOR READ ONLY.
- A cursor in the open state designates a result table and a position relative to the rows of that table. The table is the result table specified by the SELECT statement of the cursor.
- A cursor is *deletable* if each of the following conditions is true:
  - Each FROM clause of the outer fullselect identifies only one base table or deletable view (cannot identify a nested or common table expression or a nickname) without use of the OUTER clause
  - The outer fullselect does not include a VALUES clause
  - The outer fullselect does not include a GROUP BY clause or HAVING clause
  - The outer fullselect does not include column functions in the select list
  - The outer fullselect does not include SET operations (UNION, EXCEPT, or INTERSECT) with the exception of UNION ALL
  - The outer fullselect does not contain a FOR SYSTEM_TIME period specification.
  - The select list of the outer fullselect does not include DISTINCT
  - The outer fullselect does not include an ORDER BY clause (even if the ORDER BY clause is nested in a view), and the FOR UPDATE clause has not been specified
  - The select-statement does not include a FOR READ ONLY clause
  - The FROM clause of the outer fullselect does not include a *data-change-table-reference*
  - One or more of the following conditions is true:
    - The FOR UPDATE clause is specified
    - The cursor is statically defined, unless the STATICREADONLY bind option is YES
    - The LANGLEVEL bind option is MIA or SQL92E

  A column in the select list of the outer fullselect associated with a cursor is *updatable* if each of the following conditions is true:

  - The cursor is deletable
  - The column resolves to a column of the base table
  - The LANGLEVEL bind option is MIA, SQL92E or the select-statement includes the FOR UPDATE clause (the column must be specified explicitly or implicitly in the FOR UPDATE clause)

  A cursor is *read-only* if it is not deletable.

  A cursor is *ambiguous* if each of the following conditions is true:

  - The select-statement is dynamically prepared
  - The select-statement does not include either the FOR READ ONLY clause or the FOR UPDATE clause
  - The LANGLEVEL bind option is SAA1
  - The cursor otherwise satisfies the conditions of a deletable cursor

  An ambiguous cursor is considered read-only if the BLOCKING bind option is ALL, otherwise it is considered updatable.

- Cursors in procedures that are called by application programs written using CLI can be used to define result sets that are returned directly to the client application. Cursors in SQL procedures can also be returned to a calling SQL procedure only if they are defined using the WITH RETURN clause.
- Cursors declared in routines that are invoked directly or indirectly from a cursor declared WITH HOLD, do not inherit the WITH HOLD option. Thus, unless the cursor in the routine is explicitly defined WITH HOLD, a COMMIT in the application will close it.

  Consider the following application and two UDFs:

```
Application:

   DECLARE APPCUR CURSOR WITH HOLD FOR SELECT UDF1() ...
   OPEN APPCUR
   FETCH APPCUR ...
   COMMIT

UDF1:

   DECLARE UDF1CUR CURSOR FOR SELECT UDF2() ...
   OPEN UDF1CUR
   FETCH UDF1CUR ...

UDF2:

   DECLARE UDF2CUR CURSOR WITH HOLD FOR SELECT UDF2() ...
   OPEN UDF2CUR
   FETCH UDF2CUR ...
```

  After the application fetches cursor APPCUR, all three cursors are open. When the application issues the COMMIT statement, APPCUR remains open, because it was declared WITH HOLD. In UDF1, however, the cursor UDF1CUR is closed, because it was not defined with the WITH HOLD option. When the cursor UDF1CUR is closed, all routine invocations in the corresponding select-statement complete (receiving a final call, if so defined). UDF2 completes, which causes UDF2CUR to close.

Examples

Example 1: The DECLARE CURSOR statement associates the cursor name C1 with the results of the SELECT.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DEPARTMENT
  WHERE ADMRDEPT = 'A00';
```

Example 2: Assume that the EMPLOYEE table has been altered to add a generated column, WEEKLYPAY, that calculates the weekly pay based on the yearly salary. Declare a cursor to retrieve the system-generated column value from a row to be inserted.

```
EXEC SQL DECLARE C2 CURSOR FOR
  SELECT E.WEEKLYPAY
  FROM NEW TABLE
    (INSERT INTO EMPLOYEE
     (EMPNO, FIRSTNME, MIDINIT, LASTNAME, EDLEVEL, SALARY)
     VALUES('000420', 'Peter', 'U', 'Bender', 16, 31842) AS E;
```

DECLARE GLOBAL TEMPORARY TABLE

The DECLARE GLOBAL TEMPORARY TABLE statement defines a temporary table for the current session.

The declared temporary table description does not appear in the system catalog. It is not persistent and cannot be shared with other sessions. Each session that defines a declared global temporary table of the same name has its own unique description of the temporary table. When the session terminates, the rows of the table are deleted, and the description of the temporary table is dropped.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

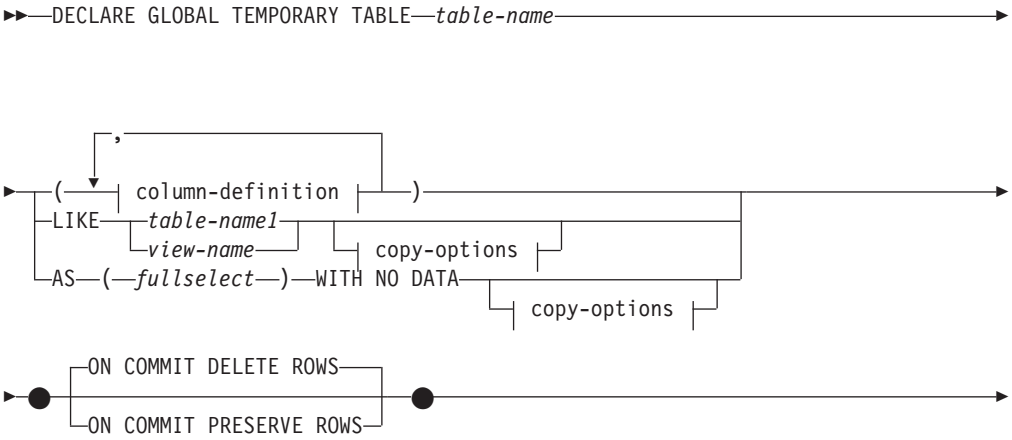
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

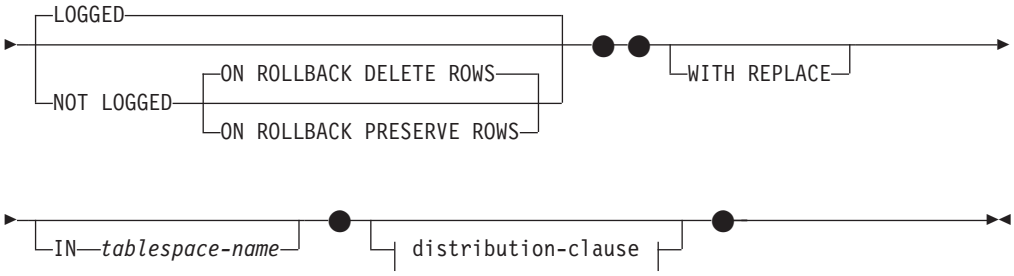
- USE privilege on the USER TEMPORARY table space
- DBADM authority
- SYSADM authority
- SYSCTRL authority

When defining a table using LIKE or a fullselect, the privileges held by the authorization ID of the statement must also include at least one of the following authorities on each identified table or view:

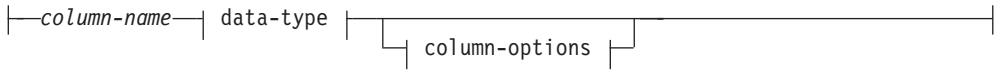
- SELECT privilege on the table or view
- CONTROL privilege on the table or view
- DATAACCESS authority

Syntax



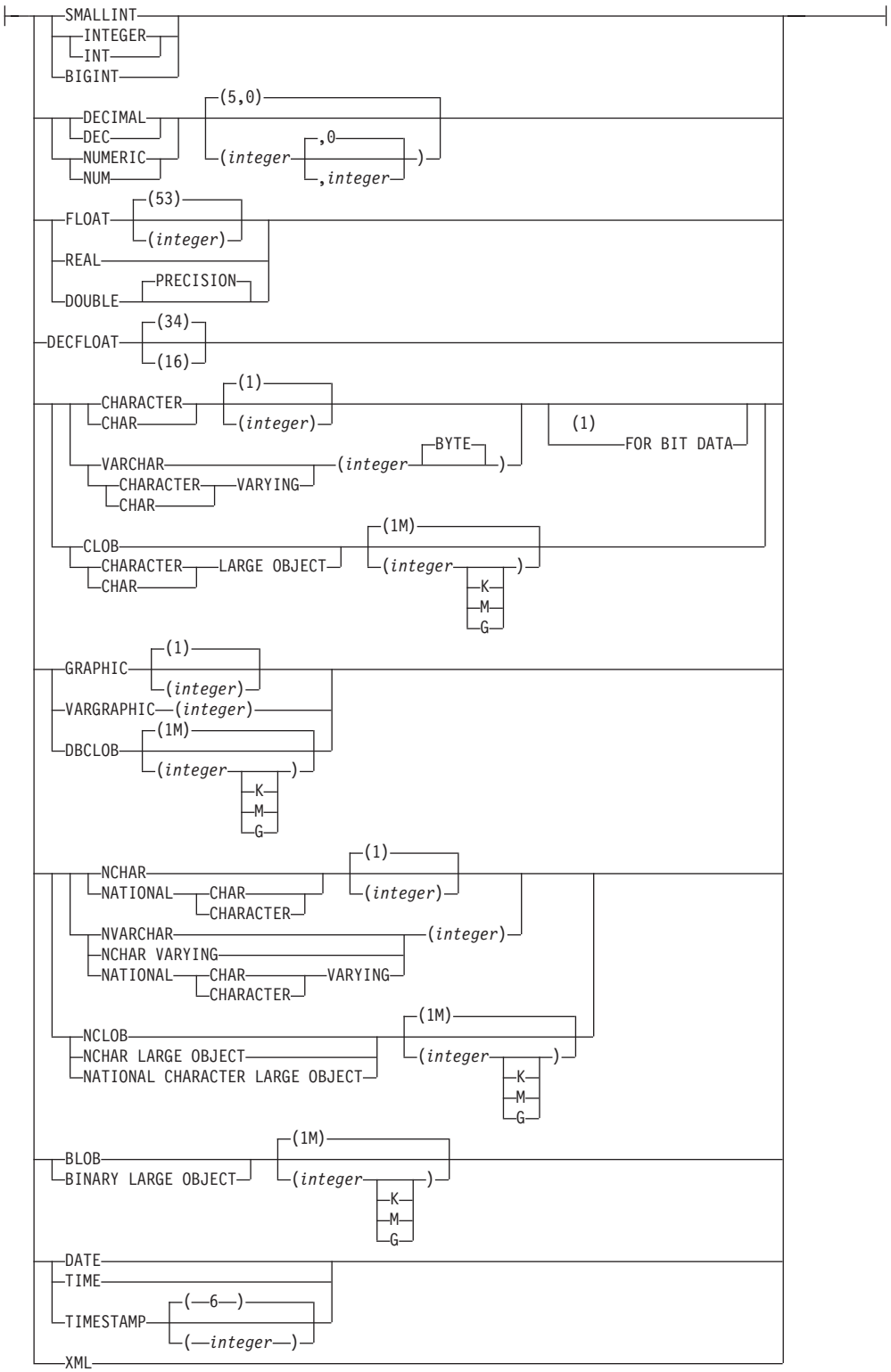


column-definition:

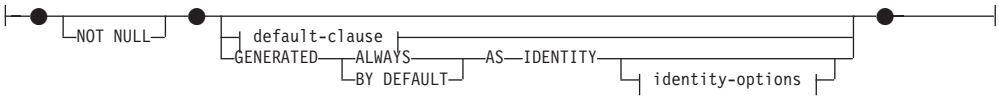


data-type:

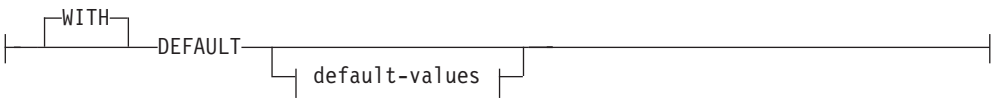
built-in-type:



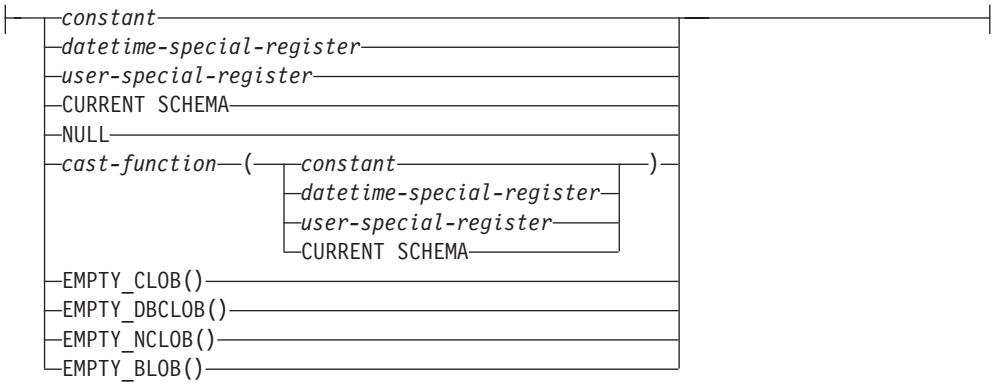
column-options:



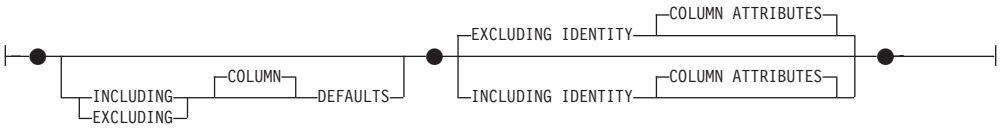
default-clause:



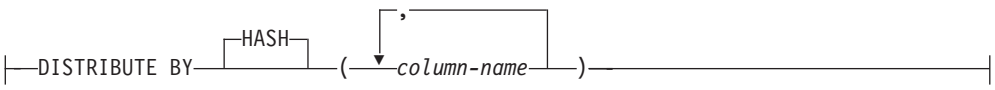
default-values:



copy-options:



distribution-clause:



Notes:

1 The FOR BIT DATA clause can be specified in any order with the other column constraints that follow.

Description

table-name

Names the temporary table. The qualifier, if specified explicitly, must be SESSION, otherwise an error is returned (SQLSTATE 428EK). If the qualifier is not specified, SESSION is implicitly assigned.

Each session that defines a declared temporary table with the same *table-name* has its own unique description of that declared temporary table. The WITH REPLACE clause must be specified if *table-name* identifies a declared temporary table that already exists in the session (SQLSTATE 42710).

It is possible that a table, view, alias, or nickname already exists in the catalog, with the same name and the schema name SESSION. In this case:

- A declared temporary table *table-name* may still be defined without any error or warning

- Any references to SESSION.*table-name* will resolve to the declared temporary table rather than the SESSION.*table-name* already defined in the catalog.

column-definition
Defines the attributes of a column of the temporary table.

column-name
Names a column of the table. The name cannot be qualified, and the same name cannot be used for more than one column of the table (SQLSTATE 42711).

A table may have the following attributes:

- A 4K page size with a maximum of 500 columns, where the byte counts of the columns must not be greater than 4 005.
- An 8K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 8 101.
- A 16K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 16 293.
- A 32K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 32 677.

A created temporary table cannot have a row-begin column, row-end column, or a transaction-start-ID column.

For more details, see "Row Size" in "CREATE TABLE".

data-type
Specifies the data type of the column

built-in-type
Specifies a built-in data type. See "CREATE TABLE" for a description of *built-in-type*.

A SYSPROC.DB2SECURITYLABEL data type cannot be specified for a declared temporary table.

column-options
Defines additional options related to the columns of the table.

NOT NULL
Prevents the column from containing null values. For specification of null values, see NOT NULL in "CREATE TABLE".

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in "ALTER TABLE".

If the column is based on a column of a typed table, a specific default value must be specified when defining a default. A default value cannot be specified for the object identifier column of a typed table (SQLSTATE 42997).

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

If a column is defined using a structured type, the *default-clause* cannot be specified (SQLSTATE 42842).

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

default-values
Specific types of default values that can be specified are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment
- not be a floating-point constant unless the column is defined with a floating-point data type
- be a numeric constant or a decimal floating-point special value if the data type of the column is a decimal floating-point. Floating-point constants are first interpreted as DOUBLE and then converted to decimal floating-point if the target column is DECFLOAT. For DECFLOAT(16) columns, decimal constants having precision greater than 16 digits will be rounded using the rounding modes specified by the CURRENT DECFLOAT ROUNDING MODE special register.
- not have nonzero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL may be specified within the same column definition but will result in an error on any attempt to set the column to the default value.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the

CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

EMPTY_CLOB(), EMPTY_DBCLOB(), or EMPTY_BLOB()
Specifies a zero-length string as the default for the column. The column must have the data type that corresponds to the result data type of the function.

If the value specified is not valid, an error is returned (SQLSTATE 42894).

IDENTITY and *identity-options*
For specification of identity columns, see IDENTITY and *identity-options* in "CREATE TABLE".

LIKE *table-name1* **or** *view-name* **or** *nickname*
Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name1*), view (*view-name*), or nickname (*nickname*). The name specified after LIKE must identify a table, view, or nickname that exists in the catalog or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC). A protected table cannot be specified (SQLSTATE 42962).A table that has a column defined as IMPLICITLY HIDDEN cannot be specified (SQLSTATE 560AE).

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table (including implicitly hidden columns), view, or nickname. The implicit definition depends on what is identified after LIKE.

- If a table is identified, then the implicit definition includes the column name, data type and nullability characteristic of each of the columns of *table-name1*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*. The data types of the view columns must be data types that are valid for columns of a table.
- If a nickname is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each column of *nickname*.

Column default and identity column attributes may be included or excluded, based on the *copy-attributes* clauses. The implicit definition does not include any other attributes of the identified table, view, or nickname. Thus the new table does not have any unique constraints, foreign key constraints, triggers, indexes, table partitioning keys, or distribution keys. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

When a table is identified in the LIKE clause and that table contains a ROW CHANGE TIMESTAMP column, the corresponding column of the new table inherits only the data type of the ROW CHANGE TIMESTAMP column. The new column is not considered to be a generated column.

If row or column level access control (RCAC) is enforced for *table-name1*, RCAC is not inherited by the new table.

AS (*fullselect***) WITH NO DATA**
Specifies that the columns of the table have the same name and description as the columns that would appear in the derived result table of the fullselect if the fullselect were to be executed. The use of AS (*fullselect*) is an implicit

definition of *n* columns for the declared temporary table, where *n* is the number of columns that would result from the fullselect.

The implicit definition includes the following attributes of the *n* columns (if applicable to the data type):

- Column name
- Data type, length, precision, and scale
- Nullability

The following attributes are not included (the default value and identity attributes can be included by using the *copy-options*):

- Default value
- Identity attributes
- Hidden attribute
- ROW CHANGE TIMESTAMP

The implicit definition does not include any other optional attributes of the tables or views referenced in the fullselect.

Every select list element must have a unique name (SQLSTATE 42711). The AS clause can be used in the select clause to provide unique names. The fullselect must not refer to host variables or include parameter markers. The data types of the result columns of the fullselect must be data types that are valid for columns of a table.

If row or column level access control (RCAC) is enforced for any table that is specified in *fullselect*, RCAC is not cascaded to the new table.

copy-options

These options specify whether to copy additional attributes of the source result table definition (table, view, or fullselect).

INCLUDING COLUMN DEFAULTS

Column defaults for each updatable column of the source result table definition are copied. Columns that are not updatable will not have a default defined in the corresponding column of the created table.

If LIKE *table-name1* is specified, and *table-name1* identifies a base table, created temporary table, or declared temporary table, then INCLUDING COLUMN DEFAULTS is the default.

EXCLUDING COLUMN DEFAULTS

Column defaults are not copied from the source result table definition.

This clause is the default, except when LIKE *table-name* is specified and *table-name* identifies a base table, created temporary table, or declared temporary table.

INCLUDING IDENTITY COLUMN ATTRIBUTES

If available, identity column attributes (START WITH, INCREMENT BY, and CACHE values) are copied from the source's result table definition. It is possible to copy these attributes if the element of the corresponding column in the table, view, or fullselect is the name of a column of a table, or the name of a column of a view which directly or indirectly maps to the column name of a base table or created temporary table with the identity property. In all other cases, the columns of the new temporary table will not get the identity property. For example:

- The select list of the fullselect includes multiple instances of the name of an identity column (that is, selecting the same column more than once)

- The select list of the fullselect includes multiple identity columns (that is, it involves a join)
- The identity column is included in an expression in the select list
- The fullselect includes a set operation (union, except, or intersect).

EXCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are not copied from the source result table definition.

ON COMMIT
Specifies the action taken on the global temporary table when a COMMIT operation is performed. The default is DELETE ROWS.

DELETE ROWS
All rows of the table will be deleted if no WITH HOLD cursor is open on the table.

PRESERVE ROWS
Rows of the table will be preserved.

LOGGED or NOT LOGGED
Specifies whether operations for the table are logged. The default is LOGGED.

LOGGED
Specifies that insert, update, or delete operations against the table as well as the creation or dropping of the table are to be logged.

NOT LOGGED
Specifies that insert, update, or delete operations against the table are not to be logged, but that the creation or dropping of the table is to be logged. During a ROLLBACK (or ROLLBACK TO SAVEPOINT) operation:

- If the table had been created within a unit of work (or savepoint), the table is dropped
- If the table had been dropped within a unit of work (or savepoint), the table is recreated, but without any data

ON ROLLBACK
Specifies the action that is to be taken on the not logged global temporary table when a ROLLBACK (or ROLLBACK TO SAVEPOINT) operation is performed. The default is DELETE ROWS.

DELETE ROWS
If the table data has been changed, all the rows will be deleted.

PRESERVE ROWS
Rows of the table will be preserved.

WITH REPLACE
Indicates that, in the case that a declared temporary table already exists with the specified name, the existing table is replaced with the temporary table defined by this statement (and all rows of the existing table are deleted).

When WITH REPLACE is not specified, then the name specified must not identify a declared temporary table that already exists in the current session (SQLSTATE 42710).

IN *tablespace-name*
Identifies the table space in which the declared temporary table will be instantiated. The table space must exist and be a USER TEMPORARY table space (SQLSTATE 42838), over which the authorization ID of the statement has USE privilege (SQLSTATE 42501). If this clause is not specified, a table space

for the table is determined by choosing the USER TEMPORARY table space with the smallest sufficient page size over which the authorization ID of the statement has USE privilege. When more than one table space qualifies, preference is given according to who was granted the USE privilege:

1. The authorization ID
2. A group to which the authorization ID belongs
3. PUBLIC

If more than one table space still qualifies, the final choice is made by the database manager. When no USER TEMPORARY table space qualifies, an error is raised (SQLSTATE 42727).

Determination of the table space can change when:

- Table spaces are dropped or created
- USE privileges are granted or revoked

The sufficient page size of a table is determined by either the byte count of the row or the number of columns. For more details, see "Row Size" in "CREATE TABLE".

distribution-clause
Specifies the database partitioning or the way the data is distributed across multiple database partitions.

DISTRIBUTE BY HASH (*column-name***, ...)**
Specifies the use of the default hashing function on the specified columns, called a *distribution key*, as the distribution method across database partitions. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column whose data type is BLOB, CLOB, DBCLOB, XML, distinct type based on any of these types, or structured type can be used as part of a distribution key (SQLSTATE 42962).

If this clause is not specified, and the table resides in a multiple partition database partition group with multiple database partitions, the distribution key is defined as the first column whose data type is valid for a distribution key.

If none of the columns satisfies the requirements for a default distribution key, the table is created without one. Such tables are allowed only in table spaces that are defined on single-partition database partition groups.

For tables in table spaces that are defined on single-partition database partition groups, any collection of columns with data types that are valid for a distribution key can be used to define the distribution key. If this clause is not specified, no distribution key is created.

Notes

- A user temporary table space must exist before a declared temporary table can be declared (SQLSTATE 42727).
- ***Referencing a declared temporary table:*** The description of a declared temporary table does not appear in the DB2 catalog (SYSCAT.TABLES); therefore, it is not persistent and is not shareable across database connections. This means that each session that defines a declared temporary table called *table-name* has its own possibly unique description of that declared global temporary table.

  In order to reference the declared temporary table in an SQL statement (other than the DECLARE GLOBAL TEMPORARY TABLE statement), the table must

be explicitly or implicitly qualified by the schema name SESSION. If *table-name* is not qualified by SESSION, declared temporary tables are not considered when resolving the reference.

A reference to SESSION.*table-name* in a connection that has not declared a declared temporary table by that name will attempt to resolve from persistent objects in the catalog. If no such object exists, an error occurs (SQLSTATE 42704).

- When binding a package that has static SQL statements that refer to tables implicitly or explicitly qualified by SESSION, those statements will not be bound statically. When these statements are invoked, they will be incrementally bound, regardless of the VALIDATE option chosen while binding the package. At runtime, each table reference will be resolved to a declared temporary table, if it exists, or a created temporary table, or permanent table. If none exist, an error will be raised (SQLSTATE 42704).
- ***Privileges:*** When a declared temporary table is defined, the definer of the table is granted all table privileges on the table, including the ability to drop the table. Additionally, these privileges are granted to PUBLIC. (None of the privileges are granted with the GRANT option, and none of the privileges appear in the catalog table.) This enables any SQL statement in the session to reference a declared temporary table that has already been defined in that session.
- ***Instantiation and termination:*** For the following explanations, P denotes a session and T is a declared temporary table in the session P:
  - An empty instance of T is created as a result of the DECLARE GLOBAL TEMPORARY TABLE statement that is executed in P.
  - Any SQL statement in P can make reference to T and any reference to T in P is a reference to that same instance of T.
  - If a DECLARE GLOBAL TEMPORARY TABLE statement is specified within the SQL procedure compound statement (defined by BEGIN and END), the scope of the declared temporary table is the connection, not just the compound statement, and the table is known outside of the compound statement. The table is not implicitly dropped at the END of the compound statement. A declared temporary table cannot be defined multiple times by the same name in other compound statements in that session, unless the table has been explicitly dropped.
  - Assuming that the ON COMMIT DELETE ROWS clause was specified implicitly or explicitly, then when a commit operation terminates a unit of work in P, and there is no open WITH HOLD cursor in P that is dependent on T, the commit includes the operation DELETE FROM SESSION.T.
  - When a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes a modification to SESSION.T:
    - If NOT LOGGED was specified, all rows from SESSION.T are deleted unless ON ROLLBACK PRESERVE ROWS was also specified
    - If NOT LOGGED was not specified, the changes to T are undone
  - If NOT LOGGED was specified and an INSERT, UPDATE or DELETE statement fails during execution (as opposed to a compilation error), all rows from SESSION.T are deleted.
  - When a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes the declaration of SESSION.T, then the rollback includes the operation DROP SESSION.T.
  - If a rollback operation terminates a unit of work or a savepoint in P, and that unit of work or savepoint includes the drop of a declared temporary table SESSION.T, then the rollback will undo the drop of the table. If NOT LOGGED was specified, then the table will also have been emptied.

  - When the application process that declared T terminates or disconnects from the database, T is dropped and its instantiated rows are destroyed.
  - When the connection to the server at which T was declared terminates, T is dropped and its instantiated rows are destroyed.
- ***Restrictions on the use of declared temporary tables:*** Declared temporary tables cannot:
  - Be specified in an ALTER, COMMENT, GRANT, LOCK, RENAME or REVOKE statement (SQLSTATE 42995).
  - Be referenced in an AUDIT, CREATE ALIAS, or CREATE VIEW statement (SQLSTATE 42995).
  - Be specified in referential constraints (SQLSTATE 42995).
- Data row compression is enabled for a declared temporary table. When the database manager determines that there is a performance gain, table row data including XML documents stored inline in the base table object will be compressed. However, data compression of the XML storage object of a declared temporary table is not supported.
- Index compression is enabled for indexes that are created on declared temporary tables.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - DEFINITION ONLY can be specified in place of WITH NO DATA
  - The PARTITIONING KEY clause can be specified in place of the DISTRIBUTE BY clause

  The following syntax is accepted as the default behavior:
  - CCSID ASCII
  - CCSID UNICODE

Examples

- *Example 1:* Define a declared temporary table with column definitions for an employee number, salary, bonus, and commission.

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.TEMP_EMP
   (EMPNO  CHAR(6) NOT NULL,
    SALARY DECIMAL(9, 2),
    BONUS  DECIMAL(9, 2),
    COMM   DECIMAL(9, 2)) ON COMMIT PRESERVE ROWS
```

- *Example 2:* Assume that base table USER1.EMPTAB exists and that it contains three columns, one of which is an identity column. Declare a temporary table that has the same column names and attributes (including identity attributes) as the base table.

```
DECLARE GLOBAL TEMPORARY TABLE TEMPTAB1
   LIKE USER1.EMPTAB
   INCLUDING IDENTITY
   ON COMMIT PRESERVE ROWS
```

  In this example, SESSION is used as the implicit qualifier for TEMPTAB1.

DELETE

The DELETE statement deletes rows from a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect.

Deleting a row from a nickname deletes the row from the data source object to which the nickname refers. Deleting a row from a view deletes the row from the table on which the view is based if no INSTEAD OF trigger is defined for the delete operation on this view. If such a trigger is defined, the trigger will be executed instead.

There are two forms of this statement:

- The *Searched* DELETE form is used to delete one or more rows (optionally determined by a search condition).
- The *Positioned* DELETE form is used to delete exactly one row (as determined by the current position of a cursor).

Invocation

A DELETE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

To execute either form of this statement, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- DELETE privilege on the table, view, or nickname from which rows are to be deleted
- CONTROL privilege on the table, view, or nickname from which rows are to be deleted
- DATAACCESS authority

To execute a Searched DELETE statement, the privileges held by the authorization ID of the statement must also include at least one of the following authorities for each table, view, or nickname referenced by a subquery:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of a DELETE statement includes a reference to a column of the table or view in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

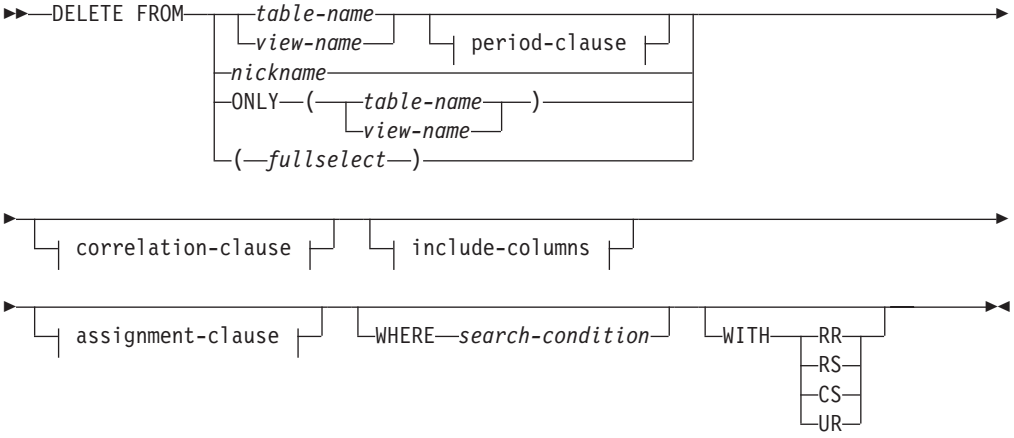
- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

Group privileges are not checked for static DELETE statements.

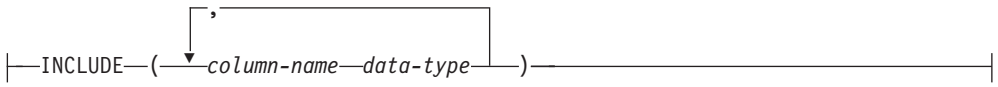
If the target of the delete operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax (searched-delete)

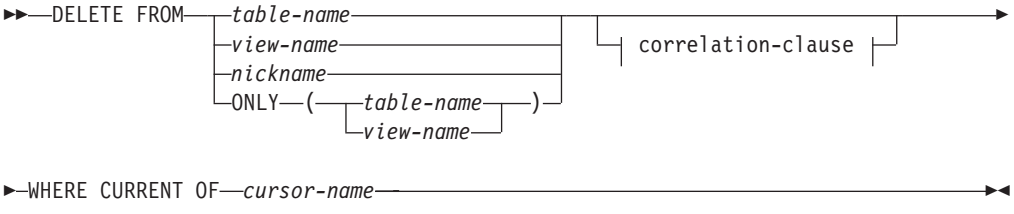


period-clause:

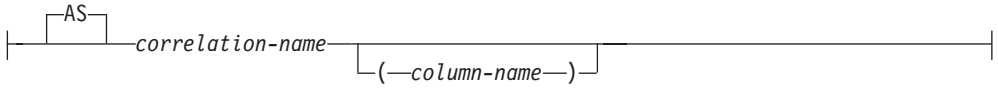
include-columns:



Syntax (positioned-delete)



correlation-clause:



Description

FROM *table-name*, *view-name*, *nickname*, **or** *(fullselect)*

Identifies the object of the delete operation. The name must identify one of the following objects:

- A table or view that exists in the catalog at the current server
- A table or view at a remote server specified using a remote-object-name

The object must not be a catalog table, a catalog view, a system-maintained materialized query table, or a read-only view.

If *table-name* is a typed table, rows of the table or any of its proper subtables may get deleted by the statement.

If *view-name* is a typed view, rows of the underlying table or underlying tables of the view's proper subviews may get deleted by the statement. If *view-name* is a regular view with an underlying table that is a typed table, rows of the typed table or any of its proper subtables may get deleted by the statement.

If the object of the delete operation is a fullselect, the fullselect must be deletable, as defined in the "Deletable views" Notes item in the description of the CREATE VIEW statement.

For additional restrictions related to temporal tables and use of a view or fullselect as the target of the delete operation, see "Considerations for a system-period temporal table" and "Considerations for an application-period temporal table" in the Notes section.

Only the columns of the specified table can be referenced in the WHERE clause. For a positioned DELETE, the associated cursor must also have specified the table or view in the FROM clause without using ONLY.

FROM ONLY (*table-name***)**

Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

FROM ONLY (*view-name***)**

Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

period-clause

Specifies that a period clause applies to the target of the delete operation.

If the target of the delete operation is a view, the following conditions apply to the view:

- The FROM clause of the outer fullselect of the view definition must include a reference, directly or indirectly, to an application-period temporal table (SQLSTATE 42724M).
- An INSTEAD OF DELETE trigger must not be defined for the view (SQLSTATE 428HY).

FOR PORTION OF BUSINESS_TIME

Specifies that the delete only applies to row values for the portion of the

period in the row that is specified by the period clause. The BUSINESS_TIME period must exist in the table (SQLSTATE 4274M). FOR PORTION OF BUSINESS_TIME must not be specified if the value of the CURRENT TEMPORAL BUSINESS_TIME special register is not NULL when the BUSTIMESENSITIVE bind option is set to YES (SQLSTATE 428HY).

FROM *value1* **TO** *value2*

Specifies that the delete applies to rows for the period specified from *value1* up to *value2*. No rows are deleted if *value1* is greater than or equal to *value2*, or if *value1* or *value2* is the null value (SQLSTATE 02000).

For the period specified with FROM *value1* TO *value2*, the BUSINESS_TIME period in a row in the target of the delete is in any of the following states:

- **Overlaps the beginning** of the specified period if the value of the begin column is less than *value1* and the value of the end column is greater than *value1*.
- **Overlaps the end**of the specified period if the value of the end column is greater than or equal to *value2* and the value of the begin column is less than *value2*.
- Is **fully contained** within the specified period if the value for the begin column for BUSINESS_TIME is greater than or equal to *value1* and the value for the corresponding end column is less than or equal to *value2*.
- Is **partially contained** in the specified period if the row overlaps the beginning of the specified period or the end of the specified period, but not both.
- **Fully overlaps** the specified period if the period in the row overlaps the beginning and end of the specified period.
- Is **not contained** in the period if both columns of BUSINESS_TIME are less than or equal to *value1* or greater than or equal to *value2*.

If the BUSINESS_TIME period in a row is not contained in the specified period, the row is not deleted. Otherwise, the delete is applied based on how the values in the columns of the BUSINESS_TIME period overlap the specified period as follows:

- If the BUSINESS_TIME period in a row is fully contained within the specified period, the row is deleted.
- If the BUSINESS_TIME period in a row is partially contained in the specified period and overlaps the beginning of the specified period:
  - The row is deleted.
  - A row is inserted using the original values from the row, except that the end column is set to *value1*.
- If the BUSINESS_TIME period in a row is partially contained in the specified period and overlaps the end of the specified period:
  - The row is deleted.
  - A row is inserted using the original values from the row, except that the begin column is set to *value2*.
- If the BUSINESS_TIME period in a row fully overlaps the specified period:
  - The row is deleted.

- A row is inserted using the original values from the row, except that the end column is set to *value1*.
- An additional row is inserted using the original values from the row, except that the begin column is set to *value2*.

value1 **and** *value2*
Each expression must return a value that has a date data type, timestamp data type, or a valid data type for a string representation of a date or timestamp (SQLSTATE 428HY). The result of each expression must be comparable to the data type of the columns of the specified period (SQLSTATE 42884). See the comparison rules described in "Assignments and comparisons".

Each expression can contain any of the following supported operands (SQLSTATE 428HY):

- Constant
- Special register
- Variable
- Scalar function whose arguments are supported operands (though user-defined functions and non-deterministic functions cannot be used)
- CAST specification where the cast operand is a supported operand
- Expression using arithmetic operators and operands

correlation-clause
Can be used within the *search-condition* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see "table-reference" in the description of "Subselect".

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the DELETE statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the DELETE statement.

column-name
Specifies a column of the intermediate result table of the DELETE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

assignment-clause
See the description of *assignment-clause* under the UPDATE statement. The same rules apply. The *include-columns* are the only columns that can be set using the *assignment-clause* (SQLSTATE 42703).

WHERE
Specifies a condition that selects the rows to be deleted. The clause can be omitted, a search condition specified, or a cursor named. If the clause is omitted, all rows of the table or view are deleted.

search-condition
Each *column-name* in the search condition, other than in a subquery must identify a column of the table or view.

The *search-condition* is applied to each row of the table, view, or nickname, and the deleted rows are those for which the result of the *search-condition* is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the *search condition* is applied to a row, and the results used in applying the *search condition*. In actuality, a subquery with no correlated references is executed once, whereas a subquery with a correlated reference may have to be executed once for each row. If a subquery refers to the object table of a DELETE statement or a dependent table with a delete rule of CASCADE or SET NULL, the subquery is completely evaluated before any rows are deleted.

CURRENT OF *cursor-name*
Identifies a cursor that is defined in a DECLARE CURSOR statement of the program. The DECLARE CURSOR statement must precede the DELETE statement.

The table, view, or nickname named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR".)

When the DELETE statement is executed, the cursor must be positioned on a row: that row is the one deleted. After the deletion, the cursor is positioned before the next row of its result table. If there is no next row, the cursor is positioned after the last row.

WITH
Specifies the isolation level used when locating the rows to be deleted.

RR Repeatable Read

RS Read Stability

CS Cursor Stability

UR Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

Rules

- ***Triggers:*** DELETE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the deleted rows. If a DELETE statement on a view causes an INSTEAD OF trigger to fire, referential integrity will be checked against the updates performed in the trigger, and not against the underlying tables of the view that caused the trigger to fire.
- ***Referential integrity:*** If the identified table or the base table of the identified view is a parent, the rows selected for delete must not have any dependents in a relationship with a delete rule of RESTRICT, and the DELETE must not cascade to descendent rows that have dependents in a relationship with a delete rule of RESTRICT.

If the delete operation is not prevented by a RESTRICT delete rule, the selected rows are deleted. Any rows that are dependents of the selected rows are also affected:

- The nullable columns of the foreign keys of any rows that are their dependents in a relationship with a delete rule of SET NULL are set to the null value.
- Any rows that are their dependents in a relationship with a delete rule of CASCADE are also deleted, and the preceding rules apply, in turn, to those rows.

The delete rule of NO ACTION is checked to enforce that any non-null foreign key refers to an existing parent row after the other referential constraints have been enforced.

- ***Security policy:*** If the identified table or the base table of the identified view is protected with a security policy, the session authorization ID must have the label-based access control (LBAC) credentials that allow:
  - Write access to all protected columns (SQLSTATE 42512)
  - Read and write access to all of the rows that are selected for deletion (SQLSTATE 42519)

Notes

- If an error occurs during the execution of a multiple row DELETE, no changes are made to the database.
- Unless appropriate locks already exist, one or more exclusive locks are acquired during the execution of a successful DELETE statement. Issuing a COMMIT or ROLLBACK statement will release the locks. Until the locks are released by a commit or rollback operation, the effect of the delete operation can only be perceived by:
  - The application process that performed the deletion
  - Another application process using isolation level UR.

  The locks can prevent other application processes from performing operations on the table.
- If an application process deletes a row on which any of its cursors are positioned, those cursors are positioned before the next row of their result table. Let C be a cursor that is positioned before row R (as a result of an OPEN, a DELETE through C, a DELETE through some other cursor, or a searched DELETE). In the presence of INSERT, UPDATE, and DELETE operations that affect the base table from which R is derived, the next FETCH operation referencing C does not necessarily position C on R. For example, the operation can position C on R', where R' is a new row that is now the next row of the result table.
- SQLERRD(3) in the SQLCA shows the number of rows that qualified for the delete operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) in the SQLCA shows the number of rows affected by referential constraints and by triggered statements. It includes rows that were deleted as a result of a CASCADE delete rule and rows in which foreign keys were set to the null value as the result of a SET NULL delete rule. With regards to triggered statements, it includes the number of rows that were inserted, updated, or deleted.
- If an error occurs that prevents deleting all rows matching the search condition and all operations required by existing referential constraints, no changes are made to the table and the error is returned.

- For nicknames, the external server option `iud_app_svpt_enforce` poses an additional limitation. Refer to the Federated documentation for more information.
- For some data sources, the SQLCODE -20190 may be returned on a delete against a nickname because of potential data inconsistency. Refer to the Federated documentation for more information.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - The FROM keyword can be omitted.
- **Considerations for a system-period temporal table:** The target of the DELETE statement must not be a fullselect that references a view in the FROM clause followed by a period specification for SYSTEM_TIME if the view is defined with the WITH CHECK OPTION and the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):
  - A subquery that references a system-period temporal table (directly or indirectly)
  - An invocation of an SQL routine that has a package associated with it
  - An invocation of an external routine with a data access indication other than NO SQL

  If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, an underlying target of the UPDATE statement must not be a system-period temporal table (SQLSTATE 51046), and the target of the DELETE statement must not be a view defined with the WITH CHECK OPTION if the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):
  - A subquery that references a system-period temporal table (directly or indirectly)
  - An invocation of an SQL routine that has a package associated with it
  - An invocation of an external routine with a data access indication other than NO SQL

  If the DELETE statement has a search condition containing a correlated subquery that references historical rows (explicitly referencing the name of the history table name or implicitly through the use of a period specification in the FROM clause), the deleted rows that are stored as historical rows are potentially visible for delete operations for the rows subsequently processed for the statement.

  The mass delete algorithm is not used for a DELETE statement for a table defined as a system-period temporal table that does not contain a search condition.
- **Considerations for a history table:** When a row of a system-period temporal table is deleted, a historical copy of the row is inserted into the corresponding history table and the end timestamp of the historical row is captured in the form of a system determined value that corresponds to the time of the data change operation. The database manager assigns the value that is generated using a reading of the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to the row begin or transaction start-ID column in a table, or a row in a system-period temporal table is deleted. The database manager ensures uniqueness of the generated values for an end column in a history table across transactions. The timestamp value might be adjusted to ensure that rows inserted into the history table have the end timestamp value greater than the begin timestamp value which can

happen when a conflicting transaction is updating the same row in the system-period temporal table (SQLSTATE 01695). The database configuration parameter **systime_period_adj** must be set to Yes for this adjustment in the timestamp value to occur otherwise and error is returned (SQLSTATE 57062).

For a delete operation, the adjustment only affects the value for the end column in the history table that corresponds to the row-end column in the associated system-period temporal table. Take these adjustments into consideration on subsequent references to the table when there is a search for the transaction start time in the row-begin column and row-end column for the SYSTEM_TIME period of the associated system-period temporal table.

- **Considerations for an application-period temporal table:** The target of the DELETE statement must not be a fullselect that references a view in the FROM clause followed by a period specification for BUSINESS_TIME if the view is defined with the WITH CHECK OPTION and the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):
  - A subquery that references an application-period temporal table (directly or indirectly)
  - An invocation of an SQL routine that has a package associated with it
  - An invocation of an external routine with a data access indication other than NO SQL

  If the CURRENT TEMPORAL BUSINESS_TIME special register is set to a non-null value, the target of the DELETE statement must not be a view defined with the WITH CHECK option if the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):
  - A subquery that references an application-period temporal table (directly or indirectly)
  - An invocation of an SQL routine that has a package associated with it
  - An invocation of an external routine with a data access indication other than NO SQL

  A DELETE statement for an application-period temporal table that contains a FOR PORTION OF BUSINESS_TIME clause indicates between which two points in time that the deletes are effective. When FOR PORTION OF BUSINESS_TIME is specified and the period value for a row, specified by the values of the row-begin column and row-end column, is only partially contained in the period specified from *value1* up to *value2*, the row is deleted and one or two rows are automatically inserted to represent the portion of the row that is not deleted. New values are generated for each generated column in an application-period temporal table for each row that is automatically inserted as a result of a delete operation on the table. If a generated column is defined as part of a unique or primary key, parent key in a referential constraint, or unique index, it is possible that an automatic insert will violate a constraint or index in which case an error is returned.

  When an application-period temporal table is the target of an DELETE statement, the value in effect for the CURRENT TEMPORAL BUSINESS_TIME special register is not the null value, and the BUSTIMESENSITIVE bind option is set to YES, the following additional predicates are implicit:

```
     bt_begin <= CURRENT TEMPORAL BUSINESS_TIME
AND bt_end > CURRENT TEMPORAL BUSINESS_TIME
```

  where `bt_begin` and `bt_end` are the begin and end columns of the BUSINESS_TIME period of the target table of the DELETE statement.

- **Considerations for application-period temporal tables and triggers:** When a row is deleted and the FOR PORTION OF BUSINESS_TIME clause is specified, additional rows may be implicitly inserted to reflect any portion of the row that was not deleted. Any existing delete triggers are activated for the rows deleted, and any existing insert triggers are activated for rows that are implicitly inserted.

Examples

- *Example 1:* Delete department (DEPTNO) 'D11' from the DEPARTMENT table.

```
DELETE FROM DEPARTMENT
  WHERE DEPTNO = 'D11'
```

- *Example 2:* Delete all the departments from the DEPARTMENT table (that is, empty the table).

```
DELETE FROM DEPARTMENT
```

- *Example 3:* Delete from the EMPLOYEE table any sales rep or field rep who didn't make a sale in 1995.

```
DELETE FROM EMPLOYEE
  WHERE LASTNAME NOT IN
    (SELECT SALES_PERSON
      FROM SALES
      WHERE YEAR(SALES_DATE)=1995)
      AND JOB IN ('SALESREP','FIELDREP')
```

-

- *Example 4:* Delete all the duplicate employee rows from the EMPLOYEE table. An employee row is considered to be a duplicate if the last names match. Keep the employee row with the smallest first name in lexical order.

```
DELETE FROM
  (SELECT ROWNUMBER() OVER (PARTITION BY LASTNAME ORDER BY FIRSTNME)
    FROM EMPLOYEE) AS E(RN)
    WHERE RN > 1
```

DESCRIBE

The DESCRIBE statement obtains information about an object.

There are two types of information that can be obtained with this statement. Each of these is described separately.

- Input parameter markers of a prepared statement. Gets information about the input parameter markers in a prepared statement. This information is put into a descriptor.
- The output of a prepared statement. Gets information about a prepared statement or information about the select list columns in a prepared SELECT statement. This information is put into a descriptor.

DESCRIBE INPUT

The DESCRIBE INPUT statement obtains information about the input parameter markers of a prepared statement.

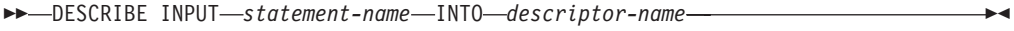
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax



Description

statement-name
Identifies the prepared statement. When the DESCRIBE INPUT statement is executed, the name must identify a statement that has been prepared by the application process at the current server.

For a CALL statement, the information returned describes the input parameters, defined as IN or INOUT, of the procedure. Input parameter markers are always considered nullable, regardless of usage.

INTO *descriptor-name*
Identifies an SQL descriptor area (SQLDA). Before the DESCRIBE INPUT statement is executed, the following variable in the SQLDA must be set:

SQLN Specifies the number of SQLVAR occurrences provided in the SQLDA. SQLN must be set to a value greater than or equal to zero before the DESCRIBE INPUT statement is executed.

When the DESCRIBE INPUT statement is executed, the database manager assigns values to the variables of the SQLDA as follows:

SQLDAID
The first 6 bytes are set to 'SQLDA ' (that is, 5 letters followed by the space character).

The seventh byte, defined as SQLDOUBLED, is set based on the parameter markers described:

- If the SQLDA contains two SQLVAR entries for every input parameter, the seventh byte is set to '2'. This technique is used to accommodate LOB or structured type input parameters.
- Otherwise, the seventh byte is set to the space character.

The seventh byte is set to the space character if there is not enough room in the SQLDA to contain the description of all input parameter markers.

The eighth byte is set to the space character.

SQLDABC
Length of the SQLDA in bytes.

SQLD The number of IN and INOUT parameters of the procedure.

SQLVAR

If the value of SQLD is 0, or greater than the value of SQLN, no values are assigned to occurrences of SQLVAR.

If the value of SQLD is *n*, where *n* is greater than 0 but less than or equal to the value of SQLN, values are assigned to the first *n* occurrences of SQLVAR. The values describe parameter markers for the input parameters of the procedure. The first occurrence of SQLVAR describes the first input parameter marker, the second occurrence of SQLVAR describes the second input parameter marker, and so on.

Base SQLVAR

SQLTYPE
A code showing the data type of the parameter and whether or not it can contain null values.

SQLLEN
A length value depending on the data type of the parameter. SQLLEN is 0 for LOB data types.

SQLNAME
The sqlname is derived as follows:

- If the SQLVAR corresponds to a parameter marker that is in the parameter list of a procedure and is not part of an expression, sqlname contains the name of the parameter if one was specified on the CREATE PROCEDURE statement.
- If the SQLVAR corresponds to a named parameter marker, sqlname contains the name of the parameter marker.
- Otherwise, sqlname contains an ASCII numeric literal value that represents the SQLVAR's position within the SQLDA.

Secondary SQLVAR

These variables are only used if the number of SQLVAR entries are doubled to accommodate LOB, distinct type, structured type, or reference type parameters.

SQLLONGLEN
The length attribute of a BLOB, CLOB, or DBCLOB parameter.

SQLDATATYPE_NAME
For any user-defined type (distinct or structured) parameter, the database manager sets this to the fully qualified user-defined type name. For a reference type parameter, the database manager sets this to the fully qualified user-defined type name of the target type of the reference. Otherwise, schema name is SYSIBM and the type name is the name in the TYPENAME column of the SYSCAT.DATATYPES catalog view.

Notes

- **Preparing the SQLDA:** Before the DESCRIBE INPUT statement is executed, the SQLDA must be allocated and the value of SQLN must be set to a value greater than or equal to zero to indicate how many occurrences of SQLVAR are provided in the SQLDA. Enough storage must be allocated to contain SQLN occurrences. To obtain the description of the input parameter markers in the prepared statement, the number of occurrences of SQLVAR must not be less than the number of input parameter markers. Furthermore, if the input parameter

markers include LOBs or structured types, the number of occurrences of SQLVAR should be two times the number of input parameter markers.

- Code page conversions between extended UNIX code (EUC) code pages and DBCS code pages, or between Unicode and non-Unicode code pages, can result in expansion or contraction of character lengths.
- If a structured type is being selected, but no FROM SQL transform is defined (either because no TRANSFORM GROUP was specified using the CURRENT DEFAULT TRANSFORM GROUP special register (SQLSTATE 428EM), or because the named group does not have a FROM SQL transform function defined (SQLSTATE 42744), an error is returned.
- **Allocating the SQLDA:** Three of the possible ways to allocate the SQLDA are as follows:

  First Technique: Allocate an SQLDA with enough occurrences of SQLVAR to accommodate any select list that the application will have to process. If the table contains any LOB, distinct type, structured type, or reference type columns, the number of SQLVARs should be double the maximum number of columns; otherwise the number should be the same as the maximum number of columns. Having done the allocation, the application can use this SQLDA repeatedly.

  This technique uses a large amount of storage that is never deallocated, even when most of this storage is not used for a particular select list.

  Second Technique: Repeat the following two steps for every processed select list:

  1. Execute a DESCRIBE INPUT statement with an SQLDA that has no occurrences of SQLVAR; that is, an SQLDA for which SQLN is zero. The value returned for SQLD is the number of columns in the result table. This is either the required number of occurrences of SQLVAR or half the required number. Because there were no SQLVAR entries, a warning with SQLSTATE 01005 will be issued. If the SQLCODE accompanying that warning is equal to one of +237, +238 or +239, the number of SQLVAR entries should be double the value returned in SQLD. (The return of these positive SQLCODEs assumes that the SQLWARN bind option setting was YES (return positive SQLCODEs). If SQLWARN was set to NO, +238 is still returned to indicate that the number of SQLVAR entries must be double the value returned in SQLD.)
  2. Allocate an SQLDA with enough occurrences of SQLVAR. Then execute the DESCRIBE statement again, using this new SQLDA.

  This technique allows better storage management than the first technique, but it doubles the number of DESCRIBE INPUT statements.

  Third Technique: Allocate an SQLDA that is large enough to handle most, and perhaps all, select lists but is also reasonably small. Execute DESCRIBE INPUT and check the SQLD value. Use the SQLD value for the number of occurrences of SQLVAR to allocate a larger SQLDA, if necessary.

  This technique is a compromise between the first two techniques. Its effectiveness depends on a good choice of size for the original SQLDA.

Example

Execute a DESCRIBE INPUT statement with an SQLDA that has enough SQLVAR occurrences to describe any number of input parameters a prepared statement might have. Assume that five parameter markers at most will need to be described and that the input data does not contain LOBs.

```
    /* STMT1_STR contains INSERT statement with VALUES clause */
EXEC SQL PREPARE STMT1_NAME FROM :STMT1_STR;
... /* code to set SQLN to 5 and to allocate the SQLDA       */
```

```
EXEC SQL DESCRIBE INPUT STMT1_NAME INTO :SQLDA;
.
.
.
```

This example uses the first technique described under “Allocating the SQLDA” in “DESCRIBE OUTPUT”.

DESCRIBE OUTPUT

The DESCRIBE OUTPUT statement obtains information about a prepared statement.

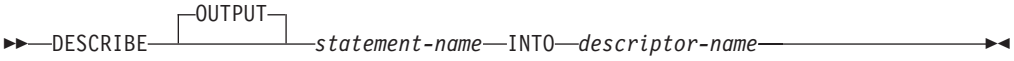
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax



Description

statement-name
Identifies the prepared statement. When the DESCRIBE OUTPUT statement is executed, the name must identify a statement that has been prepared by the application process at the current server.

If the prepared statement is a SELECT or VALUES INTO statement, the information returned describes the columns in its result table. If the prepared statement is a CALL statement, the information returned describes the output parameters, defined as OUT or INOUT, of the procedure.

INTO *descriptor-name*
Identifies an SQL descriptor area (SQLDA). Before the DESCRIBE OUTPUT statement is executed, the following variable in the SQLDA must be set:

SQLN Specifies the number of SQLVAR occurrences provided in the SQLDA. SQLN must be set to a value greater than or equal to zero before the DESCRIBE OUTPUT statement is executed.

When the DESCRIBE OUTPUT statement is executed, the database manager assigns values to the variables of the SQLDA as follows:

SQLDAID
The first 6 bytes are set to 'SQLDA ' (that is, 5 letters followed by the space character).

The seventh byte, defined as SQLDOUBLED, is set based on the results columns or parameter markers described:

- If the SQLDA contains two SQLVAR entries for every column or output parameter, the seventh byte is set to '2'. This technique is used to accommodate LOB, distinct type, structured type, or reference type columns, or output parameters.
- Otherwise, the seventh byte is set to the space character.

The seventh byte is set to the space character if there is not enough room in the SQLDA to contain the description of all result columns or output parameter markers.

The eighth byte is set to the space character.

SQLDABC
Length of the SQLDA in bytes.

SQLD If the prepared statement is a SELECT, SQLD is set to the number of columns in its result table. If the prepared statement is a CALL statement, SQLD is set to the number of OUT and INOUT parameters of the procedure. Otherwise, SQLD is set to 0.

SQLVAR
If the value of SQLD is 0, or greater than the value of SQLN, no values are assigned to occurrences of SQLVAR.

If the value of SQLD is *n*, where *n* is greater than 0 but less than or equal to the value of SQLN, values are assigned to SQLTYPE, SQLLEN, SQLNAME, SQLLONGLEN, and SQLDATATYPE_NAME for the first *n* occurrences of SQLVAR. These values describe either columns of the result table or parameter markers for the output parameters of the procedure. The first occurrence of SQLVAR describes the first column or output parameter marker, the second occurrence of SQLVAR describes the second column or output parameter marker, and so on.

Base SQLVAR

SQLTYPE
A code showing the data type of the column or parameter and whether or not it can contain null values.

SQLLEN
A length value depending on the data type of the column or parameter. SQLLEN is 0 for LOB data types.

SQLNAME
The sqlname is derived as follows:

- If the SQLVAR corresponds to a derived column for a simple column reference in the select list of a select-statement, sqlname is the name of the column.
- If the SQLVAR corresponds to a parameter marker that is in the parameter list of a procedure and is not part of an expression, sqlname contains the name of the parameter if one was specified on CREATE PROCEDURE.
- Otherwise sqlname contains an ASCII numeric literal value that represents the SQLVAR's position within the SQLDA.

Secondary SQLVAR
These variables are only used if the number of SQLVAR entries is doubled to accommodate LOB, distinct type, structured type, or reference type columns or parameters.

SQLLONGLEN
The length attribute of a BLOB, CLOB, or DBCLOB column or parameter.

SQLDATATYPE_NAME
For any user-defined type (distinct or structured) column or parameter, the database manager sets this to the fully qualified user-defined type name. For a reference type column or parameter, the database manager sets this to the fully qualified user-defined type name of the target type of the reference.

Otherwise, schema name is SYSIBM and the type name is the name in the TYPENAME column of the SYSCAT.DATATYPES catalog view.

Notes

- Before the DESCRIBE OUTPUT statement is executed, the value of SQLN must be set to indicate how many occurrences of SQLVAR are provided in the SQLDA and enough storage must be allocated to contain SQLN occurrences. For example, to obtain the description of the columns of the result table of a prepared SELECT statement, the number of occurrences of SQLVAR must not be less than the number of columns.
- If a LOB of a large size is expected, then remember that manipulating this large object will affect application memory. Given this condition, consider using locators or file reference variables. Modify the SQLDA after the DESCRIBE OUTPUT statement is executed but before allocating storage so that an SQLTYPE of SQL_TYP_xLOB is changed to SQL_TYP_xLOB_LOCATOR or SQL_TYP_xLOB_FILE with corresponding changes to other fields such as SQLLEN. Then allocate storage based on SQLTYPE and continue.
- Code page conversions between extended UNIX code (EUC) code pages and DBCS code pages, or between Unicode and non-Unicode code pages, can result in the expansion and contraction of character lengths.
- If a structured type is being selected, but no FROM SQL transform is defined (either because no TRANSFORM GROUP was specified using the CURRENT DEFAULT TRANSFORM GROUP special register (SQLSTATE 428EM), or because the named group does not have a FROM SQL transform function defined (SQLSTATE 42744), an error is returned.
- **Allocating the SQLDA:** Three of the possible ways to allocate the SQLDA are as follows:

  First Technique: Allocate an SQLDA with enough occurrences of SQLVAR to accommodate any select list that the application will have to process. If the table contains any LOB, distinct type, structured type, or reference type columns, the number of SQLVARs should be double the maximum number of columns; otherwise the number should be the same as the maximum number of columns. Having done the allocation, the application can use this SQLDA repeatedly.

  This technique uses a large amount of storage that is never deallocated, even when most of this storage is not used for a particular select list.

  Second Technique: Repeat the following two steps for every processed select list:

  1. Execute a DESCRIBE OUTPUT statement with an SQLDA that has no occurrences of SQLVAR; that is, an SQLDA for which SQLN is zero. The value returned for SQLD is the number of columns in the result table. This is either the required number of occurrences of SQLVAR or half the required number. Because there were no SQLVAR entries, a warning with SQLSTATE 01005 will be issued. If the SQLCODE accompanying that warning is equal to one of +237, +238 or +239, the number of SQLVAR entries should be double the value returned in SQLD. (The return of these positive SQLCODEs assumes that the SQLWARN bind option setting was YES (return positive SQLCODEs). If SQLWARN was set to NO, +238 is still returned to indicate that the number of SQLVAR entries must be double the value returned in SQLD.)
  2. Allocate an SQLDA with enough occurrences of SQLVAR. Then execute the DESCRIBE OUTPUT statement again, using this new SQLDA.

  This technique allows better storage management than the first technique, but it doubles the number of DESCRIBE OUTPUT statements.

Third Technique: Allocate an SQLDA that is large enough to handle most, and perhaps all, select lists but is also reasonably small. Execute DESCRIBE and check the SQLD value. Use the SQLD value for the number of occurrences of SQLVAR to allocate a larger SQLDA, if necessary.

This technique is a compromise between the first two techniques. Its effectiveness depends on a good choice of size for the original SQLDA.

- ***Considerations for implicitly hidden columns:*** A DESCRIBE OUTPUT statement returns only information about an implicitly hidden column if the column is explicitly specified as part of the SELECT list of the final result table of the query being described. If implicitly hidden columns are not part of the result table of a query, a DESCRIBE OUTPUT statement that returns information about that query will not contain information about any implicitly hidden columns.

Example

In a C program, execute a DESCRIBE OUTPUT statement with an SQLDA that has no occurrences of SQLVAR. If SQLD is greater than zero, use the value to allocate an SQLDA with the necessary number of occurrences of SQLVAR and then execute a DESCRIBE statement using that SQLDA.

```
EXEC SQL  BEGIN DECLARE SECTION;
  char  stmt1_str[200];
EXEC SQL  END DECLARE SECTION;
EXEC SQL  INCLUDE SQLDA;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

... /* code to prompt user for a query, then to generate */
    /* a select-statement in the stmt1_str            */
EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;

... /* code to set SQLN to zero and to allocate the SQLDA */
EXEC SQL  DESCRIBE STMT1_NAME INTO :sqlda;

... /* code to check that SQLD is greater than zero, to set */
    /* SQLN to SQLD, then to re-allocate the SQLDA          */
EXEC SQL  DESCRIBE STMT1_NAME INTO :sqlda;

... /* code to prepare for the use of the SQLDA            */
    /* and allocate buffers to receive the data            */
EXEC SQL  OPEN DYN_CURSOR;

... /* loop to fetch rows from result table                */
EXEC SQL  FETCH DYN_CURSOR USING DESCRIPTOR :sqlda;
.
.
.
```

DISCONNECT

The DISCONNECT statement destroys one or more connections when there is no active unit of work (that is, after a commit or rollback operation).

If a single connection is the target of the DISCONNECT statement, the connection is destroyed only if the database has participated in an existing unit of work, regardless of whether there is an active unit of work. For example, if several other databases have done work, but the target in question has not, it can still be disconnected without destroying the connection.

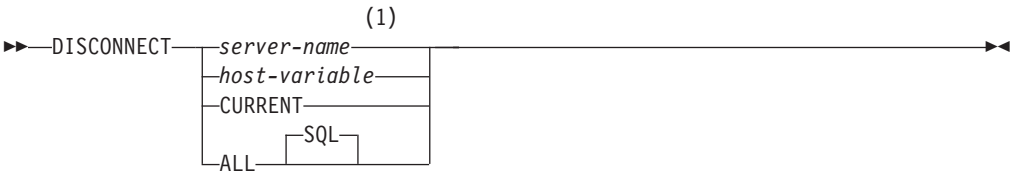
Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax



Notes:

1 Note that an application server named CURRENT or ALL can only be identified by a host variable.

Description

server-name **or** *host-variable*
Identifies the application server by the specified *server-name* or a *host-variable* which contains the *server-name*.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-aligned and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

The specified database-alias or the database-alias contained in the host variable must identify an existing connection of the application process. If the database-alias does not identify an existing connection, an error (SQLSTATE 08003) is raised.

CURRENT
Identifies the current connection of the application process. The application process must be in the connected state. If not, an error (SQLSTATE 08003) is raised.

ALL
Indicates that all existing connections of the application process are to be destroyed. An error or warning does not occur if no connections exist when the statement is executed. The optional keyword SQL is included to be consistent with the syntax of the RELEASE statement.

Rules

- Generally, the DISCONNECT statement cannot be executed while within a unit of work. If attempted, an error (SQLSTATE 25000) is raised. The exception to this rule is if a single connection is specified to be disconnected and the database has not participated in an existing unit of work. In this case, it does not matter if there is an active unit of work when the DISCONNECT statement is issued.
- The DISCONNECT statement cannot be executed at all in the Transaction Processing (TP) Monitor environment (SQLSTATE 25000). It is used when the SYNCPOINT precompiler option is set to TWOPHASE.

Notes

- If the DISCONNECT statement is successful, each identified connection is destroyed.

  If the DISCONNECT statement is unsuccessful, the connection state of the application process and the states of its connections are unchanged.
- If DISCONNECT is used to destroy the current connection, the next executed SQL statement should be CONNECT or SET CONNECTION.
- Type 1 CONNECT semantics do not preclude the use of DISCONNECT. However, though DISCONNECT CURRENT and DISCONNECT ALL can be used, they will not result in a commit operation like a CONNECT RESET statement would do.

  If *server-name* or *host-variable* is specified in the DISCONNECT statement, it must identify the current connection because Type 1 CONNECT only supports one connection at a time. Generally, DISCONNECT will fail if within a unit of work with the exception noted in "Rules".
- Resources are required to create and maintain remote connections. Thus, a remote connection that is not going to be reused should be destroyed as soon as possible.
- Connections can also be destroyed during a commit operation because the connection option is in effect. The connection option could be AUTOMATIC, CONDITIONAL, or EXPLICIT, which can be set as a precompiler option or through the SET CLIENT API at run time. For information about the specification of the DISCONNECT option, see “Distributed relational databases”.

Examples

- *Example 1:* The SQL connection to IBMSTHDB is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

  ```
  EXEC SQL DISCONNECT IBMSTHDB;
  ```
- *Example 2:* The current connection is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

  ```
  EXEC SQL DISCONNECT CURRENT;
  ```
- *Example 3:* The existing connections are no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy all the connections.

```
EXEC SQL DISCONNECT ALL;
```

DROP

The DROP statement deletes an object. Any objects that are directly or indirectly dependent on that object are either deleted or made inoperative. Whenever an object is deleted, its description is deleted from the catalog, and any packages that reference the object are invalidated.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

When dropping objects that allow two-part names, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- DROPIN privilege on the schema for the object
- Owner of the object, as recorded in the OWNER column of the catalog view for the object
- CONTROL privilege on the object (applicable only to indexes, index specifications, nicknames, packages, tables, and views)
- Owner of the user-defined type, as recorded in the OWNER column of the SYSCAT.DATATYPES catalog view (applicable only when dropping a method that is associated with a user-defined type)
- DBADM authority

When dropping a table or view hierarchy, the privileges held by the authorization ID of the statement must include one of the previously mentioned privileges for each of the tables or views in the hierarchy.

When dropping an audit policy, the privileges held by the authorization ID of the statement must include SECADM authority.

When dropping a buffer pool, database partition group, storage group, or table space, the privileges held by the authorization ID of the statement must include SYSADM or SYSCTRL authority.

When dropping a data type mapping, function mapping, server definition, or wrapper, the privileges held by the authorization ID of the statement must include DBADM authority.

When dropping an event monitor the privilege held by the authorization ID of the statement must include SQLADM or DBADM authority.

When dropping a role, the privileges held by the authorization ID of the statement must include SECADM authority.

When dropping a row permission or a column mask, the privileges held by the authorization ID of the statement must include SECADM authority.

When dropping a schema, the privileges held by the authorization ID of the statement must include DBADM authority, or be the schema owner, as recorded in the OWNER column of the SYSCAT.SCHEMATA catalog view.

When dropping a security label, a security label component, or a security policy, the privileges held by the authorization ID of the statement must include SECADM authority.

When dropping a service class, work action set, work class set, workload, threshold, or histogram template, the privileges held by the authorization ID of the statement must include WLMADM or DBADM authority.

When dropping a system-period temporal table, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

- Privileges to drop the associated history table
- Administrative authority

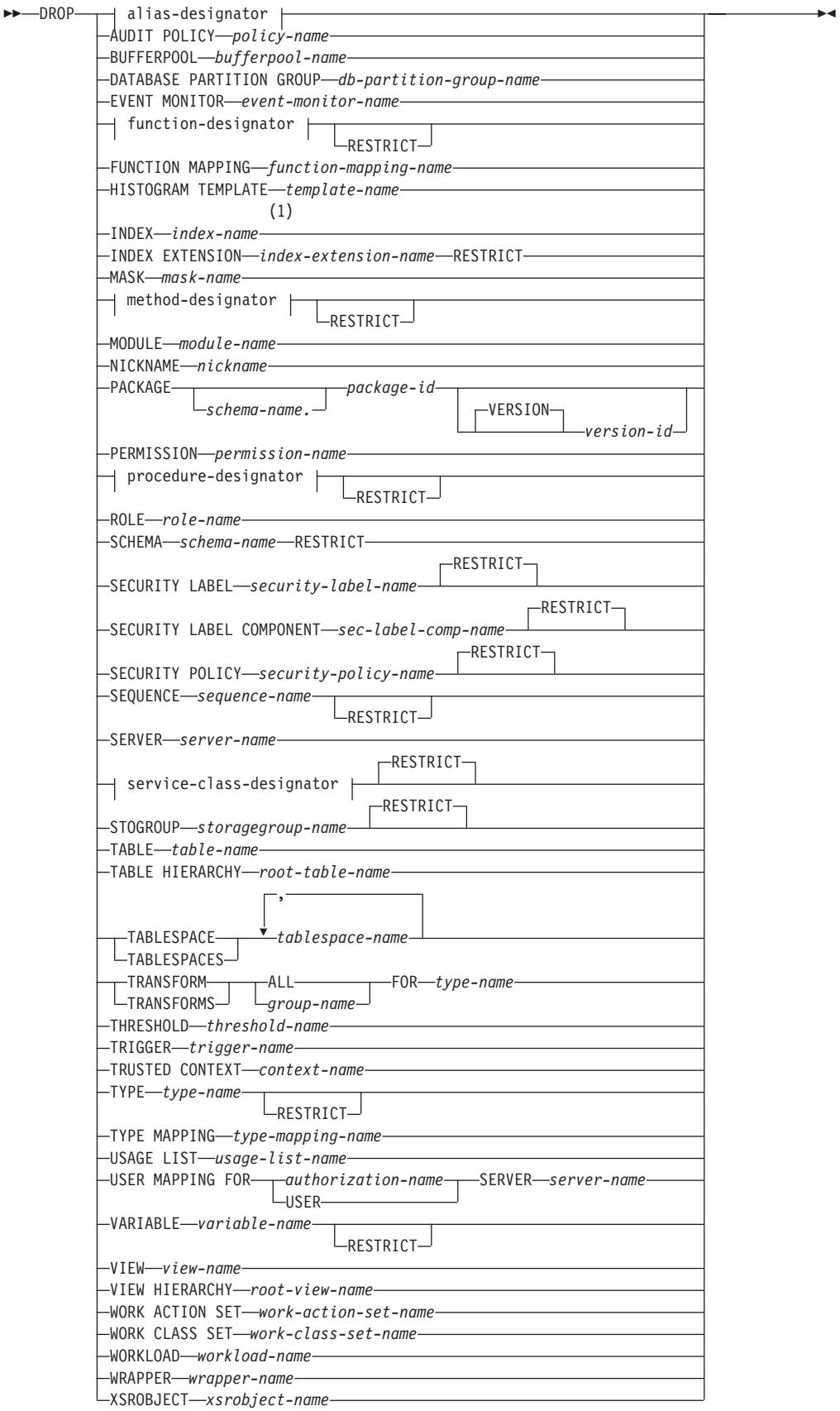
When dropping a transform, the privileges held by the authorization ID of the statement must include DBADM authority, or must be the owner of *type-name*.

When dropping a trusted context, the privileges held by the authorization ID of the statement must include SECADM authority.

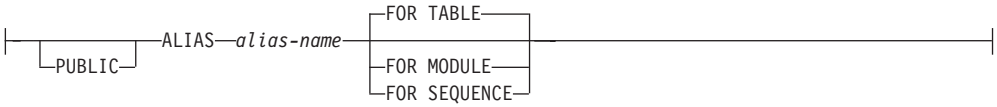
When dropping an event monitor or usage list the privilege held by the authorization ID of the statement must include SQLADM or DBADM authority.

When dropping a user mapping, the privileges held by the authorization ID of the statement must include DBADM authority, if this authorization ID is different from the federated database authorization name within the mapping. Otherwise, if the authorization ID and the authorization name match, no authorities or privileges are required.

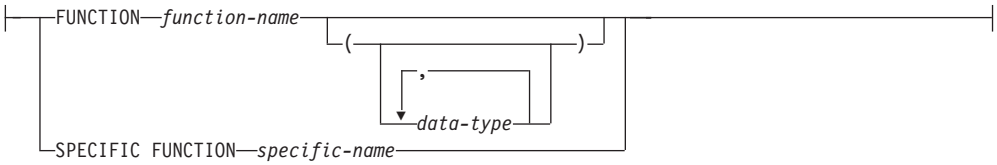
Syntax


DROP
alias-designator
AUDIT POLICY
policy-name
BUFFERPOOL
bufferpool-name
DATABASE PARTITION GROUP
db-partition-group-name
EVENT MONITOR
event-monitor-name
function-designator
RESTRICT
FUNCTION MAPPING
function-mapping-name
HISTOGRAM TEMPLATE
template-name
(1)
INDEX
index-name
INDEX EXTENSION
index-extension-name
RESTRICT
MASK
mask-name
method-designator
RESTRICT
MODULE
module-name
NICKNAME
nickname
PACKAGE
schema-name.
package-id
VERSION
version-id
PERMISSION
permission-name
procedure-designator
RESTRICT
ROLE
role-name
SCHEMA
schema-name
RESTRICT
SECURITY LABEL
security-label-name
RESTRICT
SECURITY LABEL COMPONENT
sec-label-comp-name
RESTRICT
SECURITY POLICY
security-policy-name
RESTRICT
SEQUENCE
sequence-name
RESTRICT
SERVER
server-name
service-class-designator
RESTRICT
STOGROUP
storagegroup-name
RESTRICT
TABLE
table-name
TABLE HIERARCHY
root-table-name
,
TABLESPACE
TABLESPACES
tablespace-name
TRANSFORM
TRANSFORMS
ALL
group-name
FOR
type-name
THRESHOLD
threshold-name
TRIGGER
trigger-name
TRUSTED CONTEXT
context-name
TYPE
type-name
RESTRICT
TYPE MAPPING
type-mapping-name
USAGE LIST
usage-list-name
USER MAPPING FOR
authorization-name
USER
SERVER
server-name
VARIABLE
variable-name
RESTRICT
VIEW
view-name
VIEW HIERARCHY
root-view-name
WORK ACTION SET
work-action-set-name
WORK CLASS SET
work-class-set-name
WORKLOAD
workload-name
WRAPPER
wrapper-name
XSROBJECT
xsrobject-name

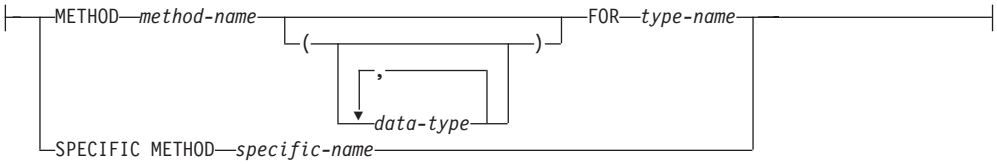
alias-designator:



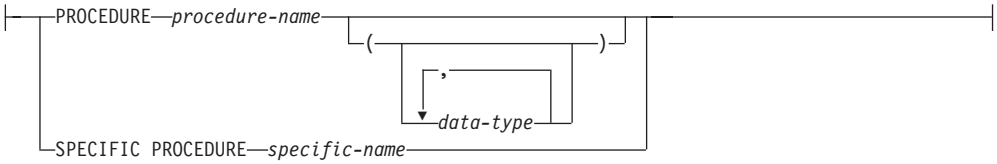
function-designator:



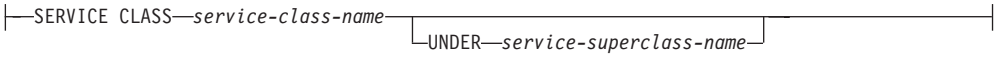
method-designator:



procedure-designator:



service-class-designator:



Notes:

1 *Index-name* can be the name of either an index or an index specification.

Description

alias-designator

ALIAS *alias-name*
Identifies the alias that is to be dropped. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). The specified alias is deleted.

FOR TABLE, FOR MODULE, or FOR SEQUENCE
Specifies the object type for the alias.

FOR TABLE
The alias is for a table, view, or nickname.

FOR MODULE
The alias is for a module.

FOR SEQUENCE
The alias is for a sequence.

All views and triggers that reference the alias are made inoperative. This includes alias references in both the ON clause of the CREATE TRIGGER statement and within the triggered SQL statements. Any materialized query table or staging table that references the alias is dropped.

If PUBLIC is specified, the *alias-name* must identify a public alias that exists at the current server (SQLSTATE 42704).

If the alias is referenced in the definition of a row permission or a column mask, the alias cannot be dropped (SQLSTATE 42893).

AUDIT POLICY *policy-name*
Identifies the audit policy that is to be dropped. The *policy-name* must identify an audit policy that exists at the current server (SQLSTATE 42704). The audit policy must not be associated with any database objects (SQLSTATE 42893). The specified audit policy is deleted from the catalog.

BUFFERPOOL *bufferpool-name*
Identifies the buffer pool that is to be dropped. The *bufferpool-name* must identify a buffer pool that is described in the catalog (SQLSTATE 42704). There can be no table spaces assigned to the buffer pool (SQLSTATE 42893). The IBMDEFAULTBP buffer pool cannot be dropped (SQLSTATE 42832). Buffer pool memory is released immediately, to be used by DB2. Disk storage may not be released until the next connection to the database.

DATABASE PARTITION GROUP *db-partition-group-name*
Identifies the database partition group that is to be dropped. The *db-partition-group-name* parameter must identify a database partition group that is described in the catalog (SQLSTATE 42704). This is a one-part name.

Dropping a database partition group drops all table spaces defined in the database partition group. All existing database objects with dependencies on the tables in the table spaces (such as packages, referential constraints, and so on) are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

IBMCATGROUP, IBMDEFAULTGROUP, and IBMTEMPGROUP database partition groups cannot be dropped (SQLSTATE 42832).

If a DROP DATABASE PARTITION GROUP statement is issued against a database partition group that is currently undergoing a data redistribution, the drop database partition group operation fails, and an error is returned (SQLSTATE 55038). However, a partially redistributed database partition group can be dropped. A database partition group can become partially redistributed if a REDISTRIBUTE DATABASE PARTITION GROUP command does not execute to completion. This can happen if it is interrupted by either an error or a FORCE APPLICATION ALL command. (For a partially redistributed database partition group, the REDISTRIBUTE_PMAP_ID in the SYSCAT.DBPARTITIONGROUPS catalog is not -1.)

EVENT MONITOR *event-monitor-name*
Identifies the event monitor that is to be dropped. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

If the identified event monitor is active, an error is returned (SQLSTATE 55034); otherwise, the event monitor is deleted. Note that if an event monitor has been previously activated using the SET EVENT MONITOR STATE statement, and the database has been deactivated and subsequently reactivated, use the SET EVENT MONITOR STATE statement to deactivate the event monitor before issuing the DROP statement.

If there are event files in the target path of a WRITE TO FILE event monitor that is being dropped, the event files are not deleted. However, if a new event monitor that specifies the same target path is created, the event files are deleted.

When dropping WRITE TO TABLE event monitors, table information is removed from the SYSCAT.EVENTTABLES catalog view, but the tables themselves are not dropped.

function-designator
Identifies an instance of a user-defined function (either a complete function or a function template) that is to be dropped. For more information, see "Function, method, and procedure designators" on page 20.

The function instance specified must be a user-defined function described in the catalog. The following functions cannot be dropped:

- A function implicitly generated by a CREATE TYPE statement (SQLSTATE 42917)
- A function that is in the SYSIBM, SYSFUN, SYSIBMADM, or the SYSPROC schema (SQLSTATE 42832)
- A function that is referenced in the definition of a row permission or a column mask (SQLSTATE 42893)

RESTRICT
The RESTRICT keyword enforces the rule that the function is not to be dropped if any of the following dependencies exists:

- Another function is sourced on the function.
- Another routine uses the function.
- A view uses the function.
- A trigger uses the function.
- A materialized query table uses the function in its definition.

The restrict rule is enforced by default for the same dependencies as in version 9.5 if the **auto_reval** database configuration parameter is set to disabled.

In this case, the following considerations apply:

- Other objects can be dependent upon a function. All such dependencies must be removed before the function can be dropped, with the exception of packages which are marked inoperative. An attempt to drop a function with such dependencies will result in an error (SQLSTATE 42893). See the "Rules" section for a list of these dependencies. If the function can be dropped, it is dropped.
- Any package dependent on the specific function being dropped is marked as inoperative. Such a package is not implicitly rebound. It must either be rebound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

It is not possible to drop a function that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

If the function is referenced in the definition of a row permission or a column mask, the function cannot be dropped (SQLSTATE 42893).

FUNCTION MAPPING *function-mapping-name*
Identifies the function mapping that is to be dropped. The *function-mapping-name* must identify a user-defined function mapping that is described in the catalog (SQLSTATE 42704). The function mapping is deleted from the database.

Default function mappings cannot be dropped, but can be disabled by using the CREATE FUNCTION MAPPING statement. Dropping a user-defined function mapping that was created to override a default function mapping reinstates the default function mapping.

Packages having a dependency on a dropped function mapping are invalidated.

HISTOGRAM TEMPLATE *template-name*
Identifies the histogram template that is to be dropped. The *template-name* must identify a histogram template that exists at the current server (SQLSTATE 42704). The *template-name* cannot be SYSDEFAULTHISTOGRAM (SQLSTATE 42832). The histogram template cannot be dropped if a service class or a work action is dependent on it (SQLSTATE 42893). The specified histogram template is deleted from the catalog.

INDEX *index-name*
Identifies the index or index specification that is to be dropped. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704). It cannot be an index that is required by the system for a primary key or unique constraint, for a replicated materialized query table, or for an XML column (SQLSTATE 42917). The specified index or index specification is deleted.

Packages having a dependency on a dropped index or index specification are invalidated.

INDEX EXTENSION *index-extension-name* **RESTRICT**
Identifies the index extension that is to be dropped. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no index can be defined that depends on this index extension definition (SQLSTATE 42893).

MASK *mask-name*
Identifies the column mask to drop. The name must identify a column mask that exists at the current server (SQLSTATE 42704).

method-designator
Identifies a method body that is to be dropped. For more information, see "Function, method, and procedure designators" on page 20. The method body specified must be a method described in the catalog (SQLSTATE 42704). Method bodies that are implicitly generated by the CREATE TYPE statement cannot be dropped.

DROP METHOD deletes the body of a method, but the method specification (signature) remains as a part of the definition of the subject type. After dropping the body of a method, the method specification can be removed from the subject type definition by ALTER TYPE DROP METHOD.

RESTRICT
The RESTRICT keyword enforces the rule that the method is not to be dropped if any of the following dependencies exists:

- A function is sourced on the method.
- Another routine uses the method.
- A view uses the method.
- A trigger uses the method.
- A materialized query table uses the method in its definition.

The restrict rule is enforced by default for the same dependencies as in version 9.5 if the **auto_reval** database configuration parameter is set to disabled.

In this case, the following considerations apply:

- Other objects can be dependent upon a method. All such dependencies must be removed before the method can be dropped, with the exception of packages which will be marked inoperative if the drop is successful. An attempt to drop a method with such dependencies will result in an error (SQLSTATE 42893). If the method can be dropped, it will be dropped.
- Any package dependent on the specific method being dropped is marked as inoperative. Such a package is not implicitly re-bound. Either it must be re-bound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

If the specific method being dropped overrides another method, all packages dependent on the overridden method - and on methods that override this method in supertypes of the specific method being dropped - are invalidated.

MODULE *module-name*
Identifies the module that is to be dropped. The *module-name* must identify a module that exists at the current server (SQLSTATE 42704). The specified module is dropped from the schema, including all module objects. All privileges on the module are also dropped.

If the module is referenced in the definition of a row permission or a column mask, the module cannot be dropped (SQLSTATE 42893).

NICKNAME *nickname*
Identifies the nickname that is to be dropped. The nickname must be listed in the catalog (SQLSTATE 42704). The nickname is deleted from the database.

All information about the columns and indexes associated with the nickname is deleted from the catalog. Any materialized query tables that are dependent on the nickname are dropped. Any index specifications that are dependent on the nickname are dropped. Any views that are dependent on the nickname are marked inoperative. Any packages that are dependent on the dropped index specifications or inoperative views are invalidated. The data source table that the nickname references is not affected.

If an SQL function or method is dependent on a nickname, that nickname cannot be dropped (SQLSTATE 42893).

PACKAGE *schema-name.package-id*
Identifies the package that is to be dropped. If a schema name is not specified, the package identifier is implicitly qualified by the default schema. The schema name and package identifier, together with the implicitly or explicitly specified version identifier, must identify a package that is described in the catalog (SQLSTATE 42704). The specified package is deleted. If the package being dropped is the only package identified by *schema-name.package-id* (that is, there are no other versions), all privileges on the package are also deleted.

VERSION *version-id*
Identifies which package version is to be dropped. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be dropped in one invocation of the DROP statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PERMISSION *permission-name*
Identifies the row permission to drop. The name must identify a row permission that exists at the current server (SQLSTATE 42704). The name must not identify the default row permission that was created implicitly by the DB2 database (SQLSTATE 42917).

procedure-designator
Identifies an instance of a procedure that is to be dropped. For more information, see "Function, method, and procedure designators" on page 20. The procedure instance specified must be a procedure described in the catalog. It is not possible to drop a procedure that is in the SYSIBM, SYSFUN, SYSIBMADM, or the SYSPROC schema (SQLSTATE 42832).

RESTRICT
The RESTRICT keyword prevents the procedure from being dropped if a trigger definition or an SQL routine definition contains a CALL identifying the procedure. The restrict rule is enforced by default for the same dependencies as in version 9.5 if the following conditions are met:

- The **auto_reval** database configuration parameter is set to disabled
- An inlined trigger definition, inlined SQL function definition, or inlined SQL method definition contains a CALL statement identifying the procedure

It is not possible to drop a procedure that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

ROLE *role-name*
Identifies the role that is to be dropped. The *role-name* must identify a role that already exists at the current server (SQLSTATE 42704). The *role-name* must not identify a role, or a role that contains *role-name*, if the role has either EXECUTE privilege on a routine or USAGE privilege on a sequence, and an SQL object other than a package is dependent on the routine or sequence (SQLSTATE 42893). The owner of the SQL object is either *authorization-name* or any user who is a member of *authorization-name*, where *authorization-name* is a role.

A DROP ROLE statement fails (SQLSTATE 42893) if any of the following conditions are true for the role to be dropped:

- A workload exists such that one of the values for the connection attribute SESSION_USER ROLE is *role-name*
- A trusted context using *role-name* exists

The specified role is deleted from the catalog.

SCHEMA *schema-name* **RESTRICT**
Identifies the particular schema to be dropped. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no objects can be defined in the specified schema for the schema to be deleted from the database (SQLSTATE 42893).

SECURITY LABEL *security-label-name*
Identifies the security label to be dropped. The name must be qualified with a security policy (SQLSTATE 42704) and must identify a security label that exists at the current server (SQLSTATE 42704).

RESTRICT
This option, which is the default, prevents the security label from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more authorization IDs currently hold the security label for read access
- One or more authorization IDs currently hold the security label for write access
- The security label is currently being used to protect one or more columns

SECURITY LABEL COMPONENT *sec-label-comp-name*
Identifies the security label component to be dropped. The *sec-label-comp-name* must identify a security label component that is described in the catalog (SQLSTATE 42704).

RESTRICT
This option, which is the default, prevents the security label component from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more security policies that include the security label component are currently defined

SECURITY POLICY *security-policy-name*
Identifies the security policy to be dropped. The *security-policy-name* must identify a security policy that exists at the current server (SQLSTATE 42704).

RESTRICT
This option, which is the default, prevents the security policy from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more tables are associated with this security policy
- One or more authorization IDs hold an exemption on one of the rules in this security policy
- One or more security labels are defined for this security policy

SEQUENCE *sequence-name*
Identifies the particular sequence that is to be dropped. The *sequence-name*, along with the implicit or explicit schema name, must identify an existing sequence at the current server. If no sequence by this name exists in the explicitly or implicitly specified schema, an error is returned (SQLSTATE 42704).

RESTRICT
The RESTRICT keyword prevents the sequence from being dropped if any of the following dependencies exist:

- A trigger exists such that a NEXT VALUE or PREVIOUS VALUE expression in the trigger body specifies the sequence (SQLSTATE 42893).

- An SQL routine exists such that a NEXT VALUE expression in the routine body specifies the sequence (SQLSTATE 42893).

The restrict rule is enforced by default for the same dependencies as in version 9.5 if the following conditions are met:

- The **auto_reval** database configuration parameter is set to disabled
- An inlined trigger definition, inlined SQL function definition, or inlined SQL method definition references the sequence

SERVER *server-name*
Identifies the data source whose definition is to be dropped from the catalog. The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The definition of the data source is deleted.

All nicknames for tables and views residing at the data source are dropped. Any index specifications dependent on these nicknames are dropped. Any user-defined function mappings, user-defined type mappings, and user mappings that are dependent on the dropped server definition are also dropped. All packages dependent on the dropped server definition, function mappings, nicknames, and index specifications are invalidated. All federated procedures that are dependent on the server definition are also dropped.

service-class-designator

SERVICE CLASS *service-class-name*
Identifies the service class to be dropped. The *service-class-name* must identify a service class that is described in the catalog (SQLSTATE 42704). To drop a service subclass, the *service-superclass-name* must be specified using the UNDER clause.

UNDER *service-superclass-name*
Specifies the service superclass of the service subclass when dropping a service subclass. The *service-superclass-name* must identify a service superclass that is described in the catalog (SQLSTATE 42704).

RESTRICT
This keyword enforces the rule that the service class is not to be dropped if any of the following dependencies exists:

- The service class is a service superclass and there is a user defined service subclass under the service class (SQLSTATE 5U031). The service subclass must first be dropped.
- The service class is a service superclass and there is a work action set mapping to the service class (SQLSTATE 5U031). The work action set must first be dropped.
- The service class is a service subclass and there is a work action mapping to the service class (SQLSTATE 5U031). The work action must first be dropped.
- The service class has a workload mapping (SQLSTATE 5U031). The workload mapping must first be removed. Remove the workload mapping by dropping the workload or altering the workload to not map to the service class.
- The service class has an associated threshold (SQLSTATE 5U031). The threshold must first be dropped.
- The service class is the target of a REMAP ACTIVITY action in a threshold (SQLSTATE 5U031). Alter the threshold to set a different service subclass as the target of the REMAP ACTIVITY action or drop the threshold.

- The service class is not disabled (SQLSTATE 5U031). The service class must first be disabled.

RESTRICT is the default behavior.

STOGROUP *storagegroup-name*

Identifies the storage group that is to be dropped; *storagegroup-name* must identify a storage group that exists at the current server (SQLSTATE 42704). This is a one-part name.

RESTRICT
The RESTRICT keyword prevents the storage group from being dropped if a table space exists that uses the storage group (SQLSTATE 42893). RESTRICT is the default behavior.

The current default storage group cannot be dropped (SQLSTATE 42893). A new default can be designated using the ALTER STOGROUP statement.

The DROP STOGROUP statement cannot be executed while a database partition server is being added (SQLSTATE 55071).

TABLE *table-name*

Identifies the base table, created temporary table, or declared temporary table that is to be dropped. The *table-name* must identify a table that is described in the catalog or, if it is a declared temporary table, the *table-name* must be qualified by the schema name SESSION and exist in the application (SQLSTATE 42704). The subtables of a typed table are dependent on their supertables. All subtables must be dropped before a supertable can be dropped (SQLSTATE 42893). The *table-name* must not identify a catalog table (SQLSTATE 42832), or a history table associated with a system-period temporal table (SQLSTATE 42893). The specified table is deleted from the database.

All indexes, primary keys, foreign keys, row permissions (including the default row permission), column masks, check constraints, materialized query tables, and staging tables that are defined on the table are dropped. All views and triggers that reference the table are made inoperative, including both the table referenced in the ON clause of the CREATE TRIGGER statement and all tables referenced within the triggered SQL statements. All packages which depend on any object dropped or marked inoperative will be invalidated. This includes packages dependent on any supertables above the subtable in the hierarchy. Any referenced columns for which the dropped table is defined as the scope of the reference become unscoped.

Packages are not dependent on declared temporary tables, and therefore are not invalidated when such a table is dropped. Packages are, however, dependent on created temporary tables, and are invalidated when such a table is dropped.

In a federated system, a remote table that was created using transparent DDL can be dropped. Dropping a remote table also drops the nickname associated with that table, and invalidates any packages that are dependent on that nickname.

When a subtable is dropped from a table hierarchy, the columns associated with the subtable are no longer accessible although they continue to be considered with respect to limits on the number of columns and size of the row. Dropping a subtable has the effect of deleting all the rows of the subtable from the supertables. This may result in activation of triggers or referential integrity constraints defined on the supertables.

When a created temporary table or declared temporary table is dropped, and its creation preceded the active unit of work or savepoint, then the table will be functionally dropped and the application will not be able to access the table. However, the table will still reserve some space in its table space and will prevent that USER TEMPORARY table space from being dropped or the database partition group of the USER TEMPORARY table space from being redistributed until the unit of work is committed or savepoint is ended. Dropping a created temporary table or declared temporary table causes the data in the table to be destroyed, regardless of whether DROP is committed or rolled back.

If *table-name* is a system-period temporal table, any associated history table and any indexes defined on the history table are also dropped. To drop a system-period temporal table, the privilege set must also contain the authorization required to drop the history table (SQLSTATE 42501).

A history table associated with a system-period temporal table cannot be explicitly dropped using the DROP statement (SQLSTATE 42893). A history table is implicitly dropped when the associated system-period temporal table is dropped.

A table cannot be dropped if it has the RESTRICT ON DROP attribute.

A newly detached table is initially inaccessible. This prevents the table from being read, modified, or dropped until the SET INTEGRITY statement can be run to incrementally refresh MQTs or to complete any processing for foreign key constraints. After the SET INTEGRITY statement executes against all dependent tables, the table is fully accessible, its detached attribute is reset, and it can be dropped.

When a table is dropped, all row permissions, including the default row permission, and column masks that are created for the table are also dropped.

If the table is referenced in the definition of a row permission or a column mask, the table cannot be dropped (SQLSTATE 42893).

TABLE HIERARCHY *root-table-name*
Identifies the typed table hierarchy that is to be dropped. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR). The typed table identified by *root-table-name* and all of its subtables are deleted from the database.

All indexes, materialized query tables, staging tables, primary keys, foreign keys, and check constraints referencing the dropped tables are dropped. All views and triggers that reference the dropped tables are made inoperative. All packages depending on any object dropped or marked inoperative will be invalidated. Any reference columns for which one of the dropped tables is defined as the scope of the reference become unscoped.

Unlike dropping a single subtable, dropping the table hierarchy does not result in the activation of delete triggers of any tables in the hierarchy nor does it log the deleted rows.

TABLESPACE or TABLESPACES *tablespace-name*
Identifies the table spaces that are to be dropped; *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704). This is a one-part name. *tablespace-name* must not identify a table space that contains a history table unless the system-period temporal table with which it is associated is also being dropped (SQLSTATE 42893).

The table spaces will not be dropped (SQLSTATE 55024) if there is any table that stores at least one of its parts in a table space being dropped, and has one or more of its parts in another table space that is not being dropped (these tables would need to be dropped first), or if any table that resides in the table space has the RESTRICT ON DROP attribute.

Objects whose names are prefixed with 'SYS' are built-in objects and, with the exception of the SYSTOOLSPACE and SYSTOOLSTMPSPACE table spaces, cannot be dropped (SQLSTATE 42832).

A SYSTEM TEMPORARY table space cannot be dropped (SQLSTATE 55026) if it is the only temporary table space that exists in the database. A USER TEMPORARY table space cannot be dropped if there is an instance of a created temporary table or a declared temporary table created in it (SQLSTATE 55039). Even if a created temporary table has been dropped, the USER TEMPORARY table space will still be considered to be in used until all instances of the created temporary table are dropped. Instances of a created temporary table are dropped when the session terminates or when the created temporary table is referenced in the session. Even if a declared temporary table has been dropped, the USER TEMPORARY table space will still be considered to be in use until the unit of work containing the DROP TABLE statement has been committed.

Dropping a table space drops all objects that are defined in the table space. All existing database objects with dependencies on the table space, such as packages, referential constraints, and so on, are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

Containers that were created by a user are not deleted. Any directories in the path of the container name that were created by the database manager during CREATE TABLESPACE execution are deleted. All containers that are below the database directory are deleted. When the DROP TABLESPACE statement is committed, the DMS file containers or SMS containers for the specified table space are deleted, if possible. If the containers cannot be deleted (because they are being kept open by another agent, for example), the files are truncated to zero length. After all connections are terminated, or the DEACTIVATE DATABASE command is issued, these zero-length files are deleted.

THRESHOLD *threshold-name*
Identifies the threshold that is to be dropped. The *threshold-name* must identify a threshold that exists at the current server (SQLSTATE 42704). This is a one-part name. Thresholds with a queue, for example TOTALSCPARTITIONCONNECTIONS and CONCURRENTDBCOORDACTIVITIES, must be disabled before they can be dropped (SQLSTATE 5U025). The specified threshold is deleted from the catalog.

TRIGGER *trigger-name*
Identifies the trigger that is to be dropped. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704). The specified trigger is deleted.

Dropping triggers causes certain packages to be marked invalid.

If *trigger-name* specifies an INSTEAD OF trigger on a view, another trigger may depend on that trigger through an update against the view.

TRANSFORM ALL FOR *type-name*
Indicates that all transforms groups defined for the user-defined data type *type-name* are to be dropped. The transform functions referenced in these

groups are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704).

If there are not transforms defined for *type-name*, an error is returned (SQLSTATE 42740).

DROP TRANSFORM is the inverse of CREATE TRANSFORM. It causes the transform functions associated with certain groups, for a given data type, to become undefined. The functions formerly associated with these groups still exist and can still be called explicitly, but they no longer have the transform property, and are no longer invoked implicitly for exchanging values with the host language environment.

The transform group is not dropped if there is a user-defined function (or method) written in a language other than SQL that has a dependency on one of the group's transform functions defined for the user-defined type *type-name* (SQLSTATE 42893). Such a function has a dependency on the transform function associated with the referenced transform group defined for type *type-name*. Packages that depend on a transform function associated with the named transform group are marked inoperative.

TRANSFORMS *group-name* **FOR** *type-name*
Indicates that the specified transform group for the user-defined data type *type-name* is to be dropped. The transform functions referenced in this group are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704), and the *group-name* must identify an existing transform group for *type-name*.

TRIGGER *trigger-name*
Identifies the trigger that is to be dropped. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704). The specified trigger is deleted.

Dropping triggers causes certain packages to be marked invalid.

If *trigger-name* specifies an INSTEAD OF trigger on a view, another trigger may depend on that trigger through an update against the view.

TRUSTED CONTEXT *context-name*
Identifies the trusted context that is to be dropped. The *context-name* must identify a trusted context that exists at the current server (SQLSTATE 42704). If the trusted context is dropped while trusted connections for this context are active, those connections remain trusted until they terminate or until the next reuse attempt. If an attempt is made to switch the user on these trusted connections, an error is returned (SQLSTATE 42517). The specified trusted context is deleted from the catalog.

TYPE *type-name*
Identifies the user-defined type to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. For a structured type, the associated reference type is also dropped. The *type-name* must identify a user-defined type described in the catalog.

RESTRICT
The type is not dropped (SQLSTATE 42893) if any of the following conditions are true:

- The type is used as the type of a column of a table or view.
- The type has a subtype.
- The type is a structured type used as the data type of a typed table or a typed view.
- The type is an attribute of another structured type.
- There exists a column of a table whose type might contain an instance of *type-name*. This can occur if *type-name* is the type of the column or is used elsewhere in the column's associated type hierarchy. More formally, for any type T, T cannot be dropped if there exists a column of a table whose type directly or indirectly uses *type-name*.
- The type is the target type of a reference-type column of a table or view, or a reference-type attribute of another structured type.
- The type, or a reference to the type, is a parameter type or a return value type of a function or method.
- The type is a parameter type or is used in the body of an SQL procedure.
- The type, or a reference to the type, is used in the body of an SQL function or method, but it is not a parameter type or a return value type.
- The type is used in a check constraint, trigger, view definition, or index extension.

If RESTRICT is not specified, the behavior is the same as RESTRICT, except for functions and methods that use the type. The restrict rule is enforced by default for the same dependencies as in version 9.5 if the **auto_reval** database configuration parameter is set to disabled.

Functions that use the type: If the user-defined type can be dropped, then for every function, F (with specific name SF), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following DROP FUNCTION statement is effectively executed:

```
DROP SPECIFIC FUNCTION SF
```

It is possible that this statement also would cascade to drop dependent functions. If all of these functions are also in the list to be dropped because of a dependency on the user-defined type, the drop of the user-defined type will succeed (otherwise it fails with SQLSTATE 42893).

Methods that use the type: If the user-defined type can be dropped, then for every method, M of type T1 (with specific name SM), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following statements are effectively executed:

```
DROP SPECIFIC METHOD SM
ALTER TYPE T1 DROP SPECIFIC METHOD SM
```

The existence of objects that are dependent on these methods may cause the DROP TYPE operation to fail.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

If the type is referenced in the definition of a row permission or a column mask, the type cannot be dropped (SQLSTATE 42893).

TYPE MAPPING *type-mapping-name*
Identifies the user-defined data type mapping to be dropped. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704). The data type mapping is deleted from the database.

No additional objects are dropped.

USAGE LIST *usage-list-name*
Identifies the usage list that is to be dropped. The *usage-list-name*, including the implicit or explicit qualifier, must identify a usage list that is described in the catalog (SQLSTATE 42704). Memory allocated for the usage list is released and is not under transactional control.

USER MAPPING FOR *authorization-name* | **USER SERVER** *server-name*
Identifies the user mapping to be dropped. This mapping associates an authorization name that is used to access the federated database with an authorization name that is used to access a data source. The first of these two authorization names is either identified by the *authorization-name* or referenced by the special register USER. The *server-name* identifies the data source that the second authorization name is used to access.

The *authorization-name* must be listed in the catalog (SQLSTATE 42704). The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The user mapping is deleted.

No additional objects are dropped.

VARIABLE *variable-name*
Identifies the global variable that is to be dropped. The *variable-name* must identify a global variable that exists at the current server (SQLSTATE 42704).

If the variable is referenced in the definition of a row permission or a column mask, the variable cannot be dropped (SQLSTATE 42893).

RESTRICT
The RESTRICT keyword prevents the global variable from being dropped if it is referenced in an SQL routine definition, trigger definition, or view definition (SQLSTATE 42893). The restrict rule is enforced by default for the same dependencies as in version 9.5 if the following conditions are met:

- The **auto_reval** database configuration parameter is set to disabled
- An inlined trigger definition, inlined SQL function definition, inlined SQL method definition, or view references the variable

VIEW *view-name*
Identifies the view that is to be dropped. The *view-name* must identify a view that is described in the catalog (SQLSTATE 42704). The subviews of a typed view are dependent on their superviews. All subviews must be dropped before a superview can be dropped (SQLSTATE 42893).

The specified view is deleted. The definition of any view or trigger that is directly or indirectly dependent on that view is marked inoperative. Any materialized query table or staging table that is dependent on any view that is marked inoperative is dropped. Any packages dependent on a view that is dropped or marked inoperative will be invalidated. This includes packages dependent on any superviews above the subview in the hierarchy. Any reference columns for which the dropped view is defined as the scope of the reference become unscoped.

If the view is referenced in the definition of a row permission or a column mask, the view cannot be dropped (SQLSTATE 42893).

VIEW HIERARCHY *root-view-name*
Identifies the typed view hierarchy that is to be dropped. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR). The typed view identified by *root-view-name* and all of its subviews are deleted from the database.

The definition of any view or trigger that is directly or indirectly dependent on any of the dropped views is marked inoperative. Any packages dependent on any view or trigger that is dropped or marked inoperative will be invalidated. Any reference columns for which a dropped view or view marked inoperative is defined as the scope of the reference become unscoped.

WORK ACTION SET *work-action-set-name*
Identifies the work action set that is to be dropped. The *work-action-set-name* must identify a work action set that exists at the current server (SQLSTATE 42704). All work actions that are contained by the *work-action-set-name* are also dropped.

WORK CLASS SET *work-class-set-name*
Identifies the work class set that is to be dropped. The *work-class-set-name* must identify a work class set that exists at the current server (SQLSTATE 42704). All work classes that are contained by the *work-class-set-name* are also dropped.

WORKLOAD *workload-name*
Identifies the workload that is to be dropped. This is a one-part name. The *workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). SYSDEFAULTUSERWORKLOAD or SYSDEFAULTADMWORKLOAD cannot be dropped (SQLSTATE 42832). A workload must be disabled and must not have active workload occurrences associated with it before it can be dropped (SQLSTATE 5U023). To drop a workload with an associated threshold (SQLSTATE 5U031), you must drop the threshold first. The specified workload is deleted from the catalog.

WRAPPER *wrapper-name*
Identifies the wrapper to be dropped. The *wrapper-name* must identify a wrapper that is described in the catalog (SQLSTATE 42704). The wrapper is deleted.

All server definitions, user-defined function mappings, and user-defined data type mappings that are dependent on the wrapper are dropped. All user-defined function mappings, nicknames, user-defined data type mappings, and user mappings that are dependent on the dropped server definitions are also dropped. Any index specifications dependent on the dropped nicknames are dropped, and any views dependent on these nicknames are marked inoperative. All packages dependent on the dropped objects and inoperative views are invalidated. All federated procedures that are dependent on the dropped server definitions are also dropped.

XSROBJECT *xsrobject-name*
Identifies the XSR object to be dropped. The *xsrobject-name* must identify an XSR object that is described in the catalog (SQLSTATE 42704).

Check constraints that reference the XSR object are dropped. All triggers and views referencing the XSR object are marked inoperative. Packages having a dependency on a dropped XSR object are invalidated.

In a partitioned database environment, you can issue this statement against an XSR object by connecting to any partition.

Rules

Dependencies: Table 31 on page 988 shows the dependencies that objects have on each other. Not all dependencies are explicitly recorded in the catalog. For example, there is no record of the constraints on which a package has dependencies. Four different types of dependencies are shown:

R Restrict semantics. The underlying object cannot be dropped as long as the object that depends on it exists.

C Cascade semantics. Dropping the underlying object causes the object that depends on it (the depending object) to be dropped as well. However, if the depending object cannot be dropped because it has a Restrict dependency on some other object, the drop of the underlying object will fail.

X Inoperative semantics. Dropping the underlying object causes the object that depends on it to become inoperative. It remains inoperative until a user takes some explicit action.

A Automatic invalidation and revalidation semantics. Dropping the underlying object causes the object that depends on it to become invalid. The database manager attempts to revalidate the invalid object.

A package used by a function or a method, or by a procedure that is called directly or indirectly from a function or method, will only be automatically revalidated if the routine is defined as MODIFIES SQL DATA. If the routine is not MODIFIES SQL DATA, an error is returned (SQLSTATE 56098).

In general, the database manager attempts to revalidate the invalid objects the next time the object is used. However, in situations when **auto_reval** is set to IMMEDIATE, the impacted dependent objects will be revalidated immediately after they become invalid. Those situations are:

- ALTER TABLE ... ALTER COLUMN
- ALTER TABLE ... DROP COLUMN
- ALTER TABLE ... RENAME COLUMN
- ALTER TYPE ... ADD ATTRIBUTE
- ALTER TYPE ... DROP ATTRIBUTE
- Any CREATE statement that specifies “OR REPLACE”

Some of the dependencies shown in Table 31 on page 988 change to “A” (Automatic Invalidation/Revalidation semantics) when the database configuration parameter **auto_reval** is set to IMMEDIATE or DEFERRED. Table 32 on page 994 summarizes the dependent objects that are impacted. Objects listed in the “Impacted Dependent Objects” column will be invalidated when the corresponding statement listed in the “Statement” column is executed.

Some DROP statement parameters and objects are not shown in Table 31 on page 988 because they would result in blank rows or columns:

- EVENT MONITOR, PACKAGE, PROCEDURE, SCHEMA, TYPE MAPPING, and USER MAPPING DROP statements do not have object dependencies.
- Alias, buffer pool, distribution key, privilege, and procedure object types do not have DROP statement dependencies.
- A DROP SERVER, DROP FUNCTION MAPPING, or DROP TYPE MAPPING statement in a given unit of work (UOW) cannot be processed under either of the following conditions:

– The statement references a single data source, and the UOW already includes a SELECT statement that references a nickname for a table or view within this data source (SQLSTATE 55006).
– The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes a SELECT statement that references a nickname for a table or view within one of these data sources (SQLSTATE 55006).

Table 31. Dependencies

	Object Type																										
Statement	**CONSTRAINT**	**FUNCTION**	**FUNCTION MAPPING**	**GLOBAL VARIABLE**	**INDEX**	**INDEX EXTENSION**	**MASK**	**METHOD**	**NICKNAME**	**PARTITION GROUP**	**PACKAGE**[31]	**PERMISSION**	**SERVER**	**SERVICE CLASS**	**TABLE**	**TABLESPACE**	**THRESHOLD**	**TRIGGER**	**TYPE**	**TYPE MAPPING**	**USAGE LIST**	**USER MAPPING**	**VIEW**	**WORK ACTION SET**	**WORK CLASS SET**	**WORKLOAD**	**XSROBJECT**
ALTER FUNCTION	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER METHOD	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER NICKNAME, altering the local name or the local type	R[33]	R	-	-	-	-	-	R	-	-	A	-	-	-	R	-	-	-	-	-	-	-	R	-	-	-	-
ALTER NICKNAME, altering a column option or a nickname option	-	-	-	-	-	-	-	-	-	-	A	-	-	-	R	-	-	-	-	-	-	-	-	-	-	-	-
ALTER NICKNAME, adding, altering, or dropping a constraint	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER PROCEDURE	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER SERVER	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TABLE ALTER COLUMN	-	A	-	A	-	-	R	-	-	-	A	-	-	-	-	-	-	A	-	-	-	-	A	-	-	-	X[34]
ALTER TABLE DROP COLUMN	C	C	-	C	C	-	R	-	-	-	-	R	-	-	-	-	-	C	-	-	-	-	C	-	-	-	X[34]
ALTER TABLE DROP CONSTRAINT	C	-	-	-	-	-	-	-	-	-	A[1]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TABLE DROP PARTITIONING KEY	-	-	-	-	-	-	-	-	-	R[20]	A[1]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TYPE ADD ATTRIBUTE	-	-	-	-	-	R	-	-	-	-	A[23]	-	-	-	R[24]	-	-	-	-	-	-	-	R[14]	-	-	-	-
ALTER TYPE ALTER METHOD	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 31. Dependencies (continued)

	Object Type																										
Statement	**CONSTRAINT**	**FUNCTION**	**FUNCTION MAPPING**	**GLOBAL VARIABLE**	**INDEX**	**INDEX EXTENSION**	**MASK**	**METHOD**	**NICKNAME**	**DBPARTITIONGROUP**	**PACKAGE**	**PERMISSION**	**SERVER**	**SERVICE CLASS**	**TABLE**	**TABLESPACE**	**THRESHOLD**	**TRIGGER**	**TYPE**	**TYPE MAPPING**	**USAGE LIST**	**USER MAPPING**	**VIEW**	**WORK ACTION SET**	**WORKLOAD**	**XSROBJECT**
ALTER TYPE DROP ATTRIBUTE	-	-	-	-	-	R	-	-	-	-	A [23]	-	-	-	R [24]	-	-	-	-	-	-	-	R [14]	-	-	-	-
ALTER TYPE ADD METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TYPE DROP METHOD	-	-	-	-	-	-	-	R [27]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CREATE METHOD	-	-	-	-	-	-	-	-	-	-	A [28]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CREATE TYPE	-	-	-	-	-	-	-	-	-	-	A [29]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP ALIAS	-	R	-	R	-	-	R	-	-	-	A [3]	R	-	-	C [3]	-	-	X [3]	-	-	-	-	X [3]	-	-	-	-
DROP BUFFERPOOL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-	-	-	-	-	-	-
DROP DATABASE PARTITION GROUP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-	-	-	-	-	-	-
DROP FUNCTION	R	R [7]	R	R	-	R	R	R [7]	-	-	X	R	-	-	R	-	-	R	-	-	-	-	R	-	-	-	-
DROP FUNCTION MAPPING	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP INDEX	R	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-	C [37]	-	R [17]	-	-	-	-
DROP INDEX EXTENSION	-	R	-	R	R	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP MASK	-	-	-	-	-	-	-	-	-	-	A [39]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP METHOD	R	R [7]	R	R	-	R	-	R	-	-	X, A [30]	-	-	-	R	-	-	R	-	-	-	-	R	-	-	-	-
DROP NICKNAME	-	R	-	R	C	-	-	R	-	-	A	-	-	-	C [11]	-	-	-	-	-	-	-	X [16]	-	-	-	-
DROP PERMISSION	-	-	-	-	-	-	-	-	-	-	A [40]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP PROCEDURE	-	R [7]	-	R	-	-	-	R [7]	-	-	A	-	-	-	-	-	-	R	-	-	-	-	-	-	-	-	-
DROP SEQUENCE	-	R	-	-	-	-	-	R	-	-	A	-	-	-	-	-	-	R	-	-	-	-	-	-	-	-	-
DROP SERVER	-	C [21]	C [19]	-	-	-	-	-	C	-	A	-	-	-	-	-	-	-	-	C [19]	-	C	-	-	-	-	-
DROP SERVICE CLASS	-	-	-	-	-	-	-	-	-	-	-	-	-	R [35]	-	-	R [35]	-	-	-	-	-	-	R [35]	-	R [35]	-

Table 31. Dependencies (continued)

	Object Type																									
Statement	**CONSTRAINT**	**FUNCTION**	**FUNCTION MAPPING**	**GLOBAL VARIABLE**	**INDEX**	**INDEX EXTENSION**	**MASK**	**METHOD**	**NICKNAME**	**DB PARTITION GROUP**	**PACKAGE**[31]	**PERMISSION**	**SERVER**	**SERVICE CLASS**	**TABLE**	**TABLESPACE**	**THRESHOLD**	**TRIGGER**	**TYPE**	**TYPE MAPPING**	**USAGE LIST**	**USER MAPPING**	**VIEW**	**WORK ACTION SET**	**WORK CLASS SET**	**WORKLOAD**	**XSROBJECT**
DROP STOGROUP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-	-	-	-	-	-	-
DROP TABLE[32]	C	R	-	R	C	-	R	-	-	-	A[9]	R	-	-	R, C[11]	-	-	X[16]	-	-	C[37]	-	X[16]	-	-	-	X[34]
DROP TABLE HIERARCHY	C	R	-	R	C	-	-	-	-	-	A[9]	-	-	-	R, C[11]	-	-	X[16]	-	-	-	-	X[16]	-	-	-	-
DROP TABLESPACE	-	-	-	-	C[6]	-	-	-	-	-	-	-	-	-	C, R[6]	-	-	-	-	-	-	-	-	-	-	-	-
DROP TRANSFORM	-	R	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP TRIGGER	-	-	-	-	-	-	-	-	-	-	A[1]	-	-	-	-	-	-	X[26]	-	-	-	-	-	-	-	-	-
DROP TYPE	R[13]	R[5]	-	R	-	R	-	-	-	-	A[12]	-	-	-	R[18]	-	-	R[13]	R[4]	-	-	-	R[14]	-	-	-	-
DROP VARIABLE	-	-	R	R	-	-	R	R	-	-	A	R	-	-	-	-	-	R	-	-	-	-	R	-	-	-	-
DROP VIEW	-	R	-	R	-	-	R	-	-	-	A[2]	R	-	-	-	-	-	X[16]	-	-	-	-	X[15]	-	-	-	-
DROP VIEW HIERARCHY	-	R	-	R	-	-	-	-	-	-	A[2]	-	-	-	-	-	-	X[16]	-	-	-	-	X[16]	-	-	-	-
DROP WORK CLASS SET	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R[36]	-	-
DROP WRAPPER	-	-	C	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-	-	C	-	-	-	-	-	-	-
DROP XSROBJECT	C	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	X	-	-	-	-	X	-	-	-	-
REVOKE a privilege[10]	-	C, R[25]	-	-	-	-	R[38]	C, R[25]	-	-	A[1]	R[38]	-	-	C, X[8]	-	-	X	-	-	-	-	X[8]	-	-	-	-

1 This dependency is implicit in depending on a table with these constraints, triggers, or a distribution key.

2 If a package has an INSERT, UPDATE, or DELETE statement acting upon a view, then the package has an insert, update or delete usage on the underlying base table of the view. In the case of UPDATE, the package has an update usage on each column of the underlying base table that is modified by the UPDATE.

If a package has a statement acting on a typed view, creating or dropping any view in the same view hierarchy will invalidate the package.

3 If a package, materialized query table, staging table, view, or trigger uses

an alias, it becomes dependent both on the alias and the object that the alias references. If the alias is in a chain, a dependency is created on each alias in the chain.

Aliases themselves are not dependent on anything. It is possible for an alias to be defined on an object that does not exist.

4 A user-defined type T can depend on another user-defined type B, if T:

- names B as the data type of an attribute
- has an attribute of REF(B)
- has B as a supertype.

5 If the user-defined type is referenced as a function parameter type or return type, then the type will be dropped and its catalog data will be maintained due to the routine parameter dependency. A value 'X' in the VALID column of the SYSCAT.DATATYPES catalog view indicates this dropped type. Its catalog data will be deleted by a DROP FUNCTION statement if the DROP FUNCTION statement also dropped the last routine parameter dependency on this type, or will be deleted by a CREATE TYPE statement with the same schema name, module name, and type name. If the user-defined type is a structured type, any methods that are associated with the type are also dropped.

6 Dropping a table space or a list of table spaces causes all the tables that are completely contained within the given table space or list to be dropped. However, if a table spans table spaces (indexes, long columns, or data partitions in different table spaces) and those table spaces are not in the list being dropped, the table spaces cannot be dropped as long as the table exists.

7 A function can depend on another specific function if the depending function names the base function in a SOURCE clause. A function or method can also depend on another specific function or method if the depending routine is written in SQL and uses the base routine in its body. An external method, or an external function with a structured type parameter or returns type will also depend on one or more transform functions.

8 Only loss of SELECT privilege will cause a materialized query table to be dropped or a view to become inoperative. If the view that is made inoperative is included in a typed view hierarchy, all of its subviews also become inoperative.

9 If a package has an INSERT, UPDATE, or DELETE statement acting on table T, then the package has an insert, update or delete usage on T. In the case of UPDATE, the package has an update usage on each column of T that is modified by the UPDATE.

If a package has a statement acting on a typed table, creating or dropping any table in the same table hierarchy will invalidate the package.

10 Dependencies do not exist at the column level because privileges on columns cannot be revoked individually.

If a package, trigger or view includes the use of OUTER(*Z*) in the FROM clause, there is a dependency on the SELECT privilege on every subtable or subview of *Z*. Similarly, if a package, trigger, or view includes the use of DEREF(*Y*) where *Y* is a reference type with a target table or view *Z*, there is a dependency on the SELECT privilege on every subtable or subview of *Z*.

[11] A materialized query table is dependent on the underlying tables or nicknames specified in the fullselect of the table definition.

Cascade semantics apply to dependent materialized query tables.

A subtable is dependent on its supertables up to the root table. A supertable cannot be dropped until all of its subtables are dropped.

A history table is dependent on the system-period temporal table with which it is associated. Cascade semantics apply to the history table when the system-period temporary table on which depends is dropped.

[12] A package can depend on structured types as a result of using the TYPE predicate or the subtype-treatment expression (TREAT *expression* AS *data-type*). The package has a dependency on the subtypes of each structured type specified in the right side of the TYPE predicate, or the right side of the TREAT expression. Dropping or creating a structured type that alters the subtypes on which the package is dependent causes invalidation.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

[13] A check constraint or trigger is dependent on a type if the type is used anywhere in the constraint or trigger. There is no dependency on the subtypes of a structured type used in a TYPE predicate within a check constraint or trigger.

[14] A view is dependent on a type if the type is used anywhere in the view definition (this includes the type of typed view). There is no dependency on the subtypes of a structured type used in a TYPE predicate within a view definition.

[15] A subview is dependent on its superview up to the root view. A superview cannot be dropped until all its subviews are dropped. Refer to [16] for additional view dependencies.

[16] A trigger or view is also dependent on the target table or target view of a dereference operation or DEREF function. A trigger or view with a FROM clause that includes OUTER(*Z*) is dependent on all the subtables or subviews of *Z* that existed at the time the trigger or view was created.

[17] A typed view can depend on the existence of a unique index to ensure the uniqueness of the object identifier column.

[18] A table may depend on a user defined data type (distinct or structured) because the type is:

- used as the type of a column
- used as the type of the table
- used as an attribute of the type of the table
- used as the target type of a reference type that is the type of a column of the table or an attribute of the type of the table
- directly or indirectly used by a type that is the column of the table.

[19] Dropping a server cascades to drop the function mappings and type mappings created for that named server.

[20] If the distribution key is defined on a table in a multiple partition database partition group, the distribution key is required.

[21] If a dependent OLE DB table function has "R" dependent objects (see DROP FUNCTION), then the server cannot be dropped.

22 An SQL function or method can depend on the objects referenced by its body.

23 When an attribute A of type TA of *type-name* T is dropped, the following DROP statements are effectively executed:

```
Mutator method: DROP METHOD A (TA) FOR T
Observer method: DROP METHOD A () FOR T
ALTER TYPE T
  DROP METHOD A(TA)
  DROP METHOD A()
```

24 A table may depend on an attribute of a user-defined structured data type in the following cases:

1. The table is a typed table that is based on *type-name* or any of its subtypes.
2. The table has an existing column of a type that directly or indirectly refers to *type-name*.

25 A REVOKE of SELECT privilege on a table or view that is used in the body of an SQL function or method body causes an attempt to drop the function or method body, if the function or method body defined no longer has the SELECT privilege. If such a function or method body is used in a view, trigger, function, or method body, it cannot be dropped, and the REVOKE is restricted as a result. Otherwise, the REVOKE cascades and drops such functions.

26 A trigger depends on an INSTEAD OF trigger when it modifies the view on which the INSTEAD OF trigger is defined, and the INSTEAD OF trigger fires.

27 A method declaration of an original method that is overridden by other methods cannot be dropped (SQLSTATE 42893).

28 If the method of the method body being created is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the method being created, are invalidated.

29 When a new subtype of an existing type is created, all packages dependent on methods that are defined in supertypes of the type being created, and that are eligible for overriding (for example, no mutators or observers), are invalidated.

30 If the specific method of the method body being dropped is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the specific method being dropped, are invalidated.

31 Cached dynamic SQL has the same semantics as packages.

32 When a remote base table is dropped using the DROP TABLE statement, both the nickname and the remote base table are dropped.

33 A primary key or unique keys that are not referenced by a foreign key do not restrict the altering of a nickname local name or local type.

34 An XSROBJECT can become inoperative for decomposition as a result of changes to a table that is associated with the XML schema for decomposition. Changes that could impact decomposition are: dropping the table or dropping a column of the table, or changing a column of the

table. The decomposition status of the XML schema can be reset by issuing an ALTER XSROBJECT statement to enable or disable decomposition for the XML schema.

35
- A service class cannot be dropped if any threshold is mapped to it (SQLSTATE 5U031).
- A service class cannot be dropped if any workload is mapped to it (SQLSTATE 5U031).
- A service superclass cannot be dropped until all of its user-defined service subclasses have been dropped (SQLSTATE 5U031).
- A service superclass cannot be dropped if any work action set is mapped to it (SQLSTATE 5U031).
- A service subclass cannot be dropped if any work action is mapped to it (SQLSTATE 5U031).

36 A work class set cannot be dropped until the work action set that is defined on it has been dropped.

37 Once the index or table is dropped, its usage list will be invalidated in the catalog. Revalidation will take place on the next activation of the list or it can be explicitly revalidated using the procedure ADMIN_REVALIDATE_DB_OBJECTS.

38 Revoking a privilege is restricted if it causes an object to be dropped or invalidated, and a permission or mask depends on it. For example, if you have a view which depends on a table, and a permission or mask that references the view, REVOKE SELECT on the table invalidates the view, but causes an error.

39 Packages are invalidated when a table on which the enabled permission is defined has row level access control activated on the table. Packages are not affected when dropping a permission that is disabled or is defined on a table with row access control deactivated.

40 Packages are invalidated when a table on which the enabled permission is defined has row level access control activated on the table. Packages are not affected when dropping a permission that is disabled or is defined on a table with row access control deactivated.

Table 32. Dependent Objects Impacted by **auto_reval**

Statement	Impacted Dependent Objects
ALTER NICKNAME (altering the local name or the local type)	Anchor Type, Function, Method, Procedure, User Defined Type, Variable, View
ALTER TABLE ALTER COLUMN	Anchor Type, Function, Method, Procedure, Trigger[4], User Defined Type, Variable, View, XSROBJECT
ALTER TABLE DROP COLUMN[2]	Anchor Type, Function, Method, Index, Procedure, Trigger[4], User Defined Type, Variable, View, XSROBJECT
ALTER TABLE RENAME COLUMN[1, 3]	Anchor Type, Function, Method, Index, Procedure, Trigger[4], User Defined Type, Variable, View, XSROBJECT
ALTER TYPE ADD ATTRIBUTE	View
ALTER TYPE DROP ATTRIBUTE	View
DROP ALIAS	Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View

Table 32. Dependent Objects Impacted by **auto_reval** *(continued)*

Statement	Impacted Dependent Objects
DROP FUNCTION (ALTER MODULE DROP FUNCTION)	Function, Function Mapping, Index Extension, Method, Procedure, Trigger, Variable, View
DROP METHOD	Function, Function Mapping, Index Extension, Method, Procedure, Trigger, Variable, View
DROP NICKNAME	Anchor Type, Function, Method, Procedure, Trigger, User Defined Type, Variable, View
DROP PROCEDURE (ALTER MODULE DROP PROCEDURE)	Function, Method, Procedure, Trigger
DROP SEQUENCE	Function, Method, Procedure, Trigger, Variable, View
DROP TABLE	Anchor Type, Function, Method, Procedure, Trigger[4], User Defined Type, Variable, View, XSROBJECT
DROP TABLE HIERARCHY	Function, Method, Procedure, Trigger, Variable, View
DROP TRIGGER	Trigger
DROP TYPE (ALTER MODULE DROP TYPE)	Anchor Type, Cursor Type, Function, Method, Procedure, Index Extension, Trigger, User Defined Type, Variable, View
DROP VARIABLE (ALTER MODULE DROP VARIABLE)	Anchor Type, Function, Function Mapping, Method, Procedure, Trigger, User Defined Type, Variable, View
DROP VIEW	Anchor Type, Function, Method, Procedure, Trigger[4], User Defined Type, Variable, View
DROP VIEW HIERARCHY	Function, Procedure, Trigger, Variable, View
DROP XSROBJECT	Trigger, View
RENAME TABLE	Anchor Type, Function, Method, Procedure, Trigger[4], User Defined Type, Variable, View, XSROBJECT
REVOKE a privilege	Function, Method, Procedure, Trigger, Variable, View
CREATE OR REPLACE ALIAS[1]	Function, Trigger, Procedure, Variable, View
CREATE OR REPLACE VIEW[1]	Anchor Type, Function, Method, Procedure, Trigger[4], User Defined Type, Variable, View
CREATE OR REPLACE FUNCTION[1]	Function, Function Mapping, Index Extension, method, Procedure, Variable, View
CREATE OR REPLACE PROCEDURE[1]	Function, Method, Procedure, Trigger
CREATE OR REPLACE NICKNAME[1]	Function, method, Procedure, Variable, View
CREATE OR REPLACE SEQUENCE[1]	Function, Method, Procedure, Trigger, Variable, View
CREATE OR REPLACE VARIABLE[1]	Function, Method, Procedure, Trigger, User Defined Type, Variable, View
CREATE OR REPLACE TRIGGER[1]	Trigger

[1] Immediate revalidation semantics apply for these statements (for the CREATE statements, only if OR REPLACE is specified) regardless of the setting of the **auto_reval** database configuration parameter.

[2] The dependent objects listed will be revalidated the next time the object is used, except for the following objects, which will be revalidated immediately as part of the statement:

- ANCHOR TYPE
- CURSOR TYPE

- VIEW (where the select list consists only of SELECT *, and does not contain any explicitly defined view columns).

For an immediate view revalidation, the list of column names for the select list will be re-established during revalidation.

3 The dependent objects listed will be revalidated the next time the object is used except for the following objects, which will be revalidated immediately as part of the statement:

- User Defined Type
- VIEW (where the select list consists only of `SELECT *`, and does not contain any explicitly defined view columns).

For an immediate view revalidation, the list of column names for the select list will be re-established during revalidation.

4 If the dependency is because the trigger is defined on the table or view, then the inoperative semantics from Table 1 continue to apply. If the dependency is because the trigger body references the table or view, then automatic invalidation and revalidation semantics apply.

The DROP DATABASE PARTITION GROUP statement might fail (SQLSTATE 55071) if an add database partition server request is either pending or in progress. This statement might also fail (SQLSTATE 55077) if a new database partition server is added online to the instance and not all applications are aware of the new database partition server.

Notes

- It is valid to drop a user-defined function while it is in use. Also, a cursor can be open over a statement which contains a reference to a user-defined function, and while this cursor is open the function can be dropped without causing the cursor fetches to fail.
- If a package which depends on a user-defined function is executing, it is not possible for another authorization ID to drop the function until the package completes its current unit of work. At that point, the function is dropped and the package becomes inoperative. The next request for this package results in an error indicating that the package must be explicitly rebound.
- The removal of a function body (this is very different from dropping the function) can occur while an application which needs the function body is executing. This may or may not cause the statement to fail, depending on whether the function body still needs to be loaded into storage by the database manager on behalf of the statement.
- In addition to the dependencies recorded for any explicitly specified UDF, the following dependencies are recorded when transforms are implicitly required:
  1. When the structured type parameter or result of a function or method requires a transform, a dependency is recorded for the function or method on the required TO SQL or FROM SQL transform function.
  2. When an SQL statement included in a package requires a transform function, a dependency is recorded for the package on the designated TO SQL or FROM SQL transform function.

  Since these describe the only circumstances under which dependencies are recorded due to implicit invocation of transforms, no objects other than functions, methods, or packages can have a dependency on implicitly invoked transform functions. On the other hand, explicit calls to transform functions (in views and triggers, for example) do result in the usual dependencies of these other types of objects on transform functions. As a result, a DROP TRANSFORM

statement may also fail due to these "explicit" type dependencies of objects on the transform(s) being dropped (SQLSTATE 42893).

- Since the dependency catalogs do not distinguish between depending on a function as a transform versus depending on a function by explicit function call, it is suggested that explicit calls to transform functions are not written. In such an instance, the transform property on the function cannot be dropped, or packages will be marked inoperative, simply because they contain explicit invocations in an SQL expression.
- System created sequences for IDENTITY columns cannot be dropped using the DROP SEQUENCE statement.
- When a sequence is dropped, all privileges on the sequence are also dropped and any packages that refer to the sequence are invalidated.
- For relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under either of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement is already issued in the same UOW against the nickname that is referenced in this statement
- For non-relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under any of the following conditions (SQLSTATE 55007):
  - A nickname referenced in this statement has a cursor open on it in the same UOW
  - A nickname referenced in this statement is already referenced by a SELECT statement in the same UOW
  - Either an INSERT, DELETE, or UPDATE statement has already been issued in the same UOW against the nickname that is referenced in this statement
- A DROP SERVER statement (SQLSTATE 55006), or a DROP FUNCTION MAPPING or DROP TYPE MAPPING statement (SQLSTATE 55007) within a given unit of work (UOW) cannot be processed under either of the following conditions:
  - The statement references a single data source, and the UOW already includes one of the following items:
    - A SELECT statement that references a nickname for a table or view within this data source
    - An open cursor on a nickname for a table or view within this data source
    - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within this data source
  - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following items:
    - A SELECT statement that references a nickname for a table or view within one of these data sources
    - An open cursor on a nickname for a table or view within one of these data sources
    - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within one of these data sources
- The DROP WORKLOAD statement does not take effect until it is committed, even for the connection that issues the statement.

- Only one of these statements can be issued by any application at a time, and only one of these statements is allowed within any one unit of work. Each statement must be followed by a COMMIT or a ROLLBACK statement before another one of these statements can be issued (SQLSTATE 5U021).
  - CREATE HISTOGRAM TEMPLATE, ALTER HISTOGRAM TEMPLATE, or DROP (HISTOGRAM TEMPLATE)
  - CREATE SERVICE CLASS, ALTER SERVICE CLASS, or DROP (SERVICE CLASS)
  - CREATE THRESHOLD, ALTER THRESHOLD, or DROP (THRESHOLD)
  - CREATE WORK ACTION, ALTER WORK ACTION, or DROP (WORK ACTION)
  - CREATE WORK CLASS, ALTER WORK CLASS, or DROP (WORK CLASS)
  - CREATE WORKLOAD, ALTER WORKLOAD, or DROP (WORKLOAD)
  - GRANT (Workload Privileges) or REVOKE (Workload Privileges)
- ***Soft invalidation:*** After the drop or change of a database object done by the following statements, active access to the dropped or changed object continues until the access is complete.
  - ALTER FUNCTION
  - ALTER MODULE ... DROP FUNCTION
  - ALTER MODULE ... DROP VARIABLE
  - ALTER TABLE ... DETACH PARTITION
  - ALTER VIEW
  - DROP ALIAS
  - DROP FUNCTION
  - DROP TRIGGER
  - DROP VARIABLE
  - DROP VIEW
  - All of the CREATE OR REPLACE statements except CREATE OR REPLACE SEQUENCE.

  This is the case when the database registry variable *DB2_DLL_SOFT_INVALID* is set to ON. When it is set to OFF, the drop or change of these objects will only complete after all active access to the object to be dropped or changed is complete.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
  - DISTINCT TYPE *type-name* can be specified in place of TYPE *type-name*
  - DATA TYPE *type-name* can be specified in place of TYPE *type-name*
  - SYNONYM can be specified in place of ALIAS
  - PROGRAM can be specified in place of PACKAGE
- **Invalidation of packages and dynamically cached statements after dropping row permissions or column masks:** If row level access control is activated on the table, dropping an enabled row permission defined for that table invalidates all packages and dynamically cached statements that reference that same table. If column level access control is activated on the table, dropping an enabled column mask defined for that table invalidates all packages and dynamically cached statements that reference that same table. There is no invalidation for dropping disabled masks or permissions.

- **Circular dependency:** Circular dependency exists in the following example:

```
CREATE PERMISSION RP1 ON T1 FOR ROWS
     WHERE C1>(SELECT MAX(C1) FROM T2)
ENFORCED FOR ALL ACCESS
ENABLE;

CREATE PERMISSION RP2 ON T2 FOR ROWS
     WHERE C1>(SELECT MAX(C1) FROM T1)
ENFORCED FOR ALL ACCESS
ENABLE
```

  The DROP TABLE T1 and DROP TABLE T2 statements fail because **RP1** depends on **T2** and **RP2** depends on **T1**. The user with the SECADM authority should drop one of the row permissions first then issue the **DROP TABLE** statement.

Examples

- *Example 1:* Drop table TDEPT.

```
DROP TABLE TDEPT
```

- *Example 2:* Drop the view VDEPT.

```
DROP VIEW VDEPT
```

- *Example 3:* The authorization ID HEDGES attempts to drop an alias.

```
DROP ALIAS A1
```

  The alias HEDGES.A1 is removed from the catalogs.

- *Example 4:* Hedges attempts to drop an alias, but specifies T1 as the alias-name, where T1 is the name of an existing table (not the name of an alias).

```
DROP ALIAS T1
```

  This statement fails (SQLSTATE 42809).

- *Example 5:* Drop the BUSINESS_OPS database partition group. To drop the database partition group, the two table spaces (ACCOUNTING and PLANS) in the database partition group must first be dropped.

```
DROP TABLESPACE ACCOUNTING
DROP TABLESPACE PLANS
DROP DATABASE PARTITION GROUP BUSINESS_OPS
```

- *Example 6:* Pellow wants to drop the CENTRE function, which he created in his PELLOW schema, using the signature to identify the function instance to be dropped.

```
DROP FUNCTION CENTRE (INT,FLOAT)
```

- *Example 7:* McBride wants to drop the FOCUS92 function, which she created in the PELLOW schema, using the specific name to identify the function instance to be dropped.

```
DROP SPECIFIC FUNCTION PELLOW.FOCUS92
```

- *Example 8:* Drop the function ATOMIC_WEIGHT from the CHEM schema, where it is known that there is only one function with that name.

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT
```

- *Example 9:* Drop the trigger SALARY_BONUS, which caused employees under a specified condition to receive a bonus to their salary.

```
DROP TRIGGER SALARY_BONUS
```

- *Example 10:* Drop the distinct data type named shoesize, if it is not currently in use.

```
DROP TYPE SHOESIZE
```

- *Example 11:* Drop the SMITHPAY event monitor.

```
DROP EVENT MONITOR SMITHPAY
```

- *Example 12:* Drop the schema from Example 2 under CREATE SCHEMA using RESTRICT. Notice that the table called PART must be dropped first.

```
DROP TABLE PART
DROP SCHEMA INVENTRY RESTRICT
```

- *Example 13:* Macdonald wants to drop the DESTROY procedure, which he created in the EIGLER schema, using the specific name found in the system catalog to identify the procedure to be dropped.

```
DROP SPECIFIC PROCEDURE  EIGLER.SQL100506102825100
```

- *Example 14:* Drop the procedure OSMOSIS from the BIOLOGY schema, where it is known that there is only one procedure with that name.

```
DROP PROCEDURE BIOLOGY.OSMOSIS
```

- *Example 15:* User SHAWN used one authorization ID to access the federated database and another to access the database at an Oracle data source called ORACLE1. A mapping was created between the two authorizations, but SHAWN no longer needs to access the data source. Drop the mapping.

```
DROP USER MAPPING FOR  SHAWN SERVER ORACLE1
```

- *Example 16:* An index of a data source table that a nickname references has been deleted. Drop the index specification that was created to let the optimizer know about this index.

```
DROP INDEX INDEXSPEC
```

- *Example 17:* Drop the MYSTRUCT1 transform group.

```
DROP TRANSFORM MYSTRUCT1 FOR POLYGON
```

- *Example 18:* Drop the method BONUS for the EMP data type in the PERSONNEL schema.

```
DROP METHOD BONUS (SALARY DECIMAL(10,2)) FOR PERSONNEL.EMP
```

- *Example 19:* Drop the sequence ORG_SEQ, with restrictions.

```
DROP SEQUENCE ORG_SEQ
```

- *Example 20:* A remote table EMPLOYEE was created in a federated system using transparent DDL. Access to the table is no longer needed. Drop the remote table EMPLOYEE.

```
DROP TABLE EMPLOYEE
```

- *Example 21:* Drop the function mapping BONUS_CALC and reinstate the default function mapping (if one exists).

```
DROP FUNCTION MAPPING BONUS_CALC
```

- *Example 22:* Drop the security label component LEVEL.

```
DROP SECURITY LABEL COMPONENT LEVEL
```

- *Example 23:* Drop the security label EMPLOYEESECLABEL of the security policy DATA_ACCESS.

```
DROP SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
```

- *Example 24:* Drop the security policy DATA_ACCESS.

```
DROP SECURITY POLICY DATA_ACCESS
```

- *Example 25:* Drop the security label component GROUPS.

```
DROP SECURITY LABEL COMPONENT GROUPS
```

- *Example 26:* Drop the XML schema EMPLOYEE located in the SQL schema HR.

```
DROP XSROBJECT HR.EMPLOYEE
```

- *Example 27:* Drop service subclass DOGSALES under service superclass PETSALES.

```
DROP SERVICE CLASS DOGSALES UNDER PETSALES
```

- *Example 28:* Drop service superclass PETSALES, which has no user-defined service subclasses. The default subclass for service class PETSALES is automatically dropped.

```
DROP SERVICE CLASS PETSALES
```

- *Example 29:* DROP permission P1.

```
DROP PERMISSION P1
```

- *Example 30:* DROP mask M1.

```
DROP MASK M1
```

- *Example 31:* Drop a storage group named TEST_SG.

```
DROP STOGROUP TEST_SG
```

- *Example 32:* Drop the usage list MON_PAYROLL

```
DROP USAGE LIST MON_PAYROLL
```

END DECLARE SECTION

The END DECLARE SECTION statement marks the end of a host variable declare section.

Invocation

This statement can only be embedded in an application program. It is not an executable statement. It must not be specified in REXX.

Authorization

None required.

Syntax

Description

The END DECLARE SECTION statement can be coded in the application program wherever declarations can appear according to the rules of the host language. It indicates the end of a host variable declaration section. A host variable section starts with a BEGIN DECLARE SECTION statement.

The BEGIN DECLARE SECTION and the END DECLARE SECTION statements must be paired and may not be nested.

Host variable declarations can be specified by using the SQL INCLUDE statement. Otherwise, a host variable declaration section must not contain any statements other than host variable declarations.

Host variables referenced in SQL statements must be declared in a host variable declare section in all host languages, other than REXX. Furthermore, the declaration of each variable must appear before the first reference to the variable.

Variables declared outside a declare section should not have the same name as variables declared within a declare section.

EXECUTE

The EXECUTE statement executes a prepared SQL statement.

Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

For each global variable used as an *expression* in the USING clause or in the expression for an *array-index*, the privileges held by the authorization ID of the statement must include one of the following authorities:

- READ privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

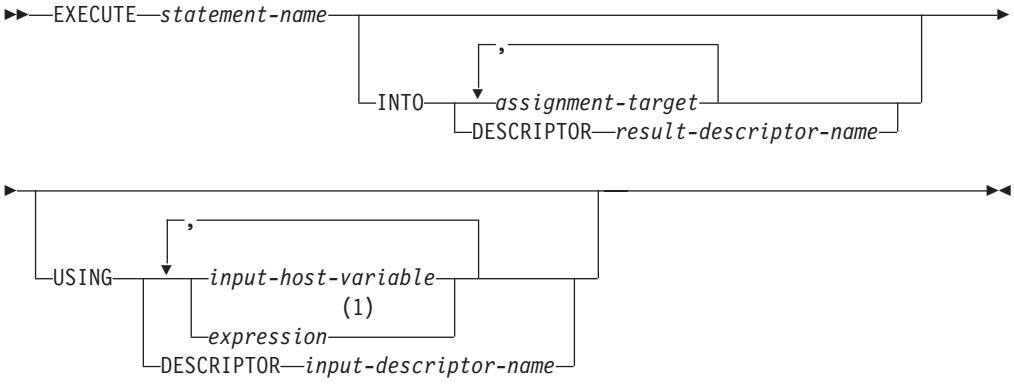
For each global variable used as an *assignment-target*, the privileges held by the authorization ID of the statement must include one of the following authorities:

- WRITE privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

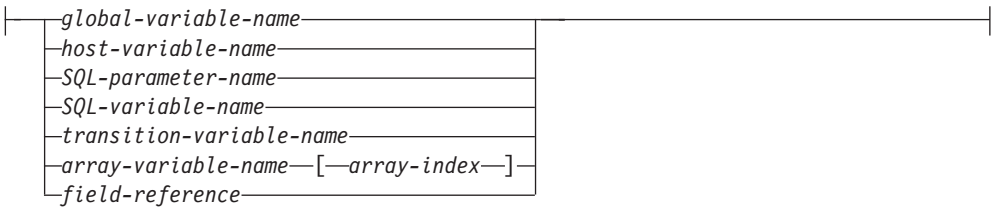
For statements where authorization checking is performed at statement execution time (DDL, GRANT, and REVOKE statements), the privileges held by the authorization ID of the statement must include those required to execute the SQL statement specified by the PREPARE statement. The authorization ID of the statement might be affected by the DYNAMICRULES bind option.

For statements where authorization checking is performed at statement preparation time (DML), no further authorization checking is performed on the SQL statement specified by the PREPARE statement.

Syntax



assignment-target:



Notes:

1 An expression other than *host-variable* can only be used when the EXECUTE statement is used within a compound SQL (compiled) statement.

Description

statement-name
Identifies the prepared statement to be executed. The *statement-name* must identify a statement that was previously prepared, and the prepared statement cannot be a SELECT statement.

INTO
Introduces a list of targets which are used to receive values from output parameter markers in the prepared statement. Each assignment to a target is made in sequence through the list. If an error occurs on any assignment, the value is not assigned to the target, and no more values are assigned to targets. Any values that have already been assigned to targets remain assigned.

For a dynamic CALL statement, parameter markers appearing in OUT and INOUT arguments to the procedure are output parameter markers. If any output parameter markers appear in the statement, the INTO clause must be specified (SQLSTATE 07007).

assignment-target
Identifies one or more targets for the assignment of output values. The first value in the result row is assigned to the first target in the list, the second value to the second target, and so on.

If the data type of an *assignment-target* is a row type, then there must be exactly one *assignment-target* specified (SQLSTATE 428HR), the number of columns must match the number of fields in the row type, and the data types of the columns of the fetched row must be assignable to the corresponding fields of the row type (SQLSTATE 42821).

If the data type of an *assignment-target* is an array element, then there must be exactly one *assignment-target* specified.

global-variable-name
Identifies the global variable that is the assignment target.

host-variable-name
Identifies the host variable that is the assignment target. For LOB output values, the target can be a regular host variable (if it is large enough), a LOB locator variable, or a LOB file reference variable.

SQL-parameter-name
Identifies the routine parameter that is the assignment target.

SQL-variable-name
Identifies the SQL variable that is the assignment target. SQL variables must be declared before they are used.

transition-variable-name
Identifies the column to be updated in the transition row. A *transition-variable-name* must identify a column in the subject table of a trigger, optionally qualified by a correlation name that identifies the new value.

array-variable-name
Identifies an SQL variable, SQL parameter, or global variable of an array type.

array-index
An expression that specifies which element in the array will be the target of the assignment. For an ordinary array, the *array-index* expression must be assignable to INTEGER (SQLSTATE 428H1) and cannot be the null value. Its value must be between 1 and the maximum cardinality defined for the array (SQLSTATE 2202E). For an associative array, the *array-index* expression must be assignable to the index data type of the associative array (SQLSTATE 428H1) and cannot be the null value.

field-reference
Identifies the field within a row type value that is the assignment target. The *field-reference* must be specified as a qualified *field-name* where the qualifier identifies the row value in which the field is defined.

DESCRIPTOR *result-descriptor-name*
Identifies an output SQLDA that must contain a valid description of host variables.

Before the EXECUTE statement is processed, the user must set the following fields in the input SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA
- SQLDABC to indicate the number of bytes of storage allocated for the SQLDA
- SQLD to indicate the number of variables used in the SQLDA when processing the statement
- SQLVAR occurrences to indicate the attributes of the variables.

The SQLDA must have enough storage to contain all SQLVAR occurrences. Therefore, the value in SQLDABC must be greater than or equal to 16 + SQLN*(N), where N is the length of an SQLVAR occurrence.

If LOB or structured data type output data must be accommodated, there must be two SQLVAR entries for every output parameter marker.

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN.

USING
Introduces a list of variables or expressions for which values are substituted for the input parameter markers in the prepared statement.

For a dynamic CALL statement, parameter markers appearing in IN and INOUT arguments to the procedure are input parameter markers. For all other dynamic statements, all the parameter markers are input parameter markers. If any input parameter markers appear in the statement, the USING clause must be specified (SQLSTATE 07004).

input-host-variable, ...
Identifies a host variable that is declared in the program in accordance with the rules for declaring host variables. The number of variables must be the same as the number of input parameter markers in the prepared statement. The *n*th variable corresponds to the *n*th parameter marker in the prepared statement. Locator variables and file reference variables, where appropriate, can be provided as the source of values for parameter markers.

expression
Identifies an expression to be used as the input for the corresponding input parameter marker in the prepared statement. An expression other than a *host-variable* can only be specified when the EXECUTE statement is issued within a compound SQL (compiled) statement.

DESCRIPTOR *input-descriptor-name*
Identifies an input SQLDA that must contain a valid description of host variables.

Before the EXECUTE statement is processed, the user must set the following fields in the input SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA
- SQLDABC to indicate the number of bytes of storage allocated for the SQLDA
- SQLD to indicate the number of variables used in the SQLDA when processing the statement
- SQLVAR occurrences to indicate the attributes of the variables.

The SQLDA must have enough storage to contain all SQLVAR occurrences. Therefore, the value in SQLDABC must be greater than or equal to 16 + SQLN*(N), where N is the length of an SQLVAR occurrence.

If LOB or structured data type input data must be accommodated, there must be two SQLVAR entries for every parameter marker.

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN.

Notes

- Before the prepared statement is executed, each input parameter marker is effectively replaced by the value of its corresponding variable or expression. For a typed parameter marker, the attributes of the target variable or expression are those specified by the CAST specification. For an untyped parameter marker, the attributes of the target variable or expression are determined according to the context of the parameter marker.

  Let V denote an input variable or expression that corresponds to parameter marker P. The value of V is assigned to the target variable for P in accordance with the rules for assigning a value to a column. Thus:

  – V must be compatible with the target.
  – If V is a string, its length must not be greater than the length attribute of the target.
  – If V is a number, the absolute value of its integral part must not be greater than the maximum absolute value of the integral part of the target.
  – If the attributes of V are not identical to the attributes of the target, the value is converted to conform to the attributes of the target.

When the prepared statement is executed, the value used in place of P is the value of the target variable for P or the result of the target expression for P. For example, if V is CHAR(6) and the target is CHAR(8), the value used in place of P is the value of V padded with two blanks.

- For a dynamic CALL statement, after the prepared statement is executed, the returned value of each OUT and INOUT argument is assigned to the assignment target corresponding to the output parameter marker used for the argument. For a typed parameter marker, the attributes of the target variable are those specified by the CAST specification. For an untyped parameter marker, the attributes of the target variable are those specified by the definition of the parameter of the procedure.

  Let V denote an output assignment target that corresponds to parameter marker P, which is used for argument A of a procedure. The value of A is assigned to V in accordance with the rules for retrieving a value from a column. Thus:

  – V must be compatible with A.
  – If V is a string, its length must not be less than the length of A, or the value of A will be truncated.
  – If V is a number, the maximum absolute value of its integral part must not be less than the absolute value of the integral part of A.
  – If the attributes of V are not identical to the attributes of A, the value of A is converted to conform to the attributes of V.

- ***Dynamic SQL statement caching:*** The information required to execute dynamic and static SQL statements is placed in the database package cache when static SQL statements are first referenced or when dynamic SQL statements are first prepared. This information stays in the package cache until it becomes invalid, the cache space is required for another statement, or the database is shut down.

  When an SQL statement is executed or prepared, the package information relevant to the application issuing the request is loaded from the system catalog into the package cache. The actual executable section for the individual SQL statement is also placed into the cache: static SQL sections are read in from the system catalog and placed in the package cache when the statement is first referenced; dynamic SQL sections are placed directly in the cache after they have been created. Dynamic SQL sections can be created by an explicit statement, such as PREPARE or EXECUTE IMMEDIATE. Once created, sections for dynamic SQL statements may be recreated by an implicit prepare of the statement by the system if the original section has been deleted for space management reasons, or has become invalid due to changes in the environment.

  Each SQL statement is cached at the database level and can be shared among applications. Static SQL statements are shared among applications using the same package; dynamic SQL statements are shared among applications using the same compilation environment, and the exact same statement text. The text of each SQL statement issued by an application is cached locally within the application for use if an implicit prepare is required. Each PREPARE statement in the application program can cache one statement. All EXECUTE IMMEDIATE statements in an application program share the same space, and only one cached statement exists for all these EXECUTE IMMEDIATE statements at a time. If the same PREPARE or any EXECUTE IMMEDIATE statement is issued multiple times with a different SQL statement each time, only the last statement will be cached for reuse. The optimal use of the cache is to issue a number of different PREPARE statements once at the start of the application, and then to issue an EXECUTE or OPEN statement as required.

  When dynamic SQL statements are cached, a statement can be reused over multiple units of work without needing to prepare the statement again, unless

the SQL statements prepared in a package are bound with the KEEPDYNAMIC NO option. The system recompiles the statement if necessary when environment changes occur.

The following events are examples of environment or data object changes that can cause cached dynamic statements to be implicitly prepared on the next PREPARE, EXECUTE, EXECUTE IMMEDIATE, or OPEN request:

- ALTER FUNCTION
- ALTER METHOD
- ALTER NICKNAME
- ALTER PROCEDURE
- ALTER SERVER
- ALTER TABLE
- ALTER TABLESPACE
- ALTER TYPE
- CREATE FUNCTION
- CREATE FUNCTION MAPPING
- CREATE INDEX
- CREATE METHOD
- CREATE PROCEDURE
- CREATE TABLE
- CREATE TEMPORARY TABLESPACE
- CREATE TRIGGER
- CREATE TYPE
- DROP (all objects)
- RUNSTATS on any table or index
- Any action that causes a view to become inoperative
- UPDATE of statistics in any system catalog table
- SET CURRENT DEGREE
- SET PATH
- SET QUERY OPTIMIZATION
- SET SCHEMA
- SET SERVER OPTION

The following list outlines the behavior that can be expected from cached dynamic SQL statements:

- *PREPARE Requests:* Subsequent preparations of the same statement do not incur the cost of compiling the statement if the section is still valid. The cost and cardinality estimates for the current cached section are returned. These values might differ from the values returned from any previous PREPARE for the same SQL statement. You do not need to issue a PREPARE statement subsequent to a COMMIT or ROLLBACK statement, unless the statement is associated with a package that was bound with KEEPDYNAMIC NO.
- *EXECUTE Requests:* EXECUTE statements may occasionally incur the cost of implicitly preparing the statement if it has become invalid since the original PREPARE. If a section is implicitly prepared, it will use the current environment and not the environment of the original PREPARE statement.
- *EXECUTE IMMEDIATE Requests:* Subsequent EXECUTE IMMEDIATE statements for the same statement will not incur the cost of compiling the statement if the section is still valid.

- *OPEN Requests:* OPEN requests for dynamically defined cursors may occasionally incur the cost of implicitly preparing the statement if it has become invalid since the original PREPARE statement. If a section is implicitly prepared, it will use the current environment and not the environment of the original PREPARE statement.
- *FETCH Requests:* No behavior changes should be expected.
- *ROLLBACK:* Only those dynamic SQL statements prepared or implicitly prepared during the unit of work affected by the rollback operation are invalidated. Inactive dynamic SQL statements associated with a package bound with `KEEPDYNAMIC NO` are removed from the application SQL context after a ROLLBACK opeation and must be explicitly prepared again before the application can execute them. Dynamic SQL statements are still cached at the database level, so a subsequent PREPARE request does not incur the cost of compiling the statement if the section is still valid.
- *COMMIT:* Dynamic SQL statements are not be invalidated, but any acquired locks are be freed. Cursors not defined with the `WITH HOLD` option are closed and their locks freed. Open cusors defined with the `WITH HOLD` option hold onto their package and section locks to protect the active section both during and after commit processing. Dynamic SQL statements bound with the `KEEPDYNAMIC NO` option are not in a prepared state after a transaction boundary and must be explicitly prepared again before the application can execute them. SELECT statements prepared for an open cursor defined with the `WITH HOLD` option remain in a prepared state until a transaction boundary is hit where the cursor is closed. Inactive dynamic SQL statements associated with a package bound with `KEEPDYNAMIC NO` are removed from the application SQL context after a commit operation and must be explicitly prepared again before the application can execute them.

If an error occurs during an implicit prepare, an error will be returned for the request causing the implicit prepare (SQLSTATE 56098).

Examples

Example 1: In this C example, an INSERT statement with parameter markers is prepared and executed. Host variables `h1 - h4` correspond to the format of TDEPT.

```
strcpy (s,"INSERT INTO TDEPT VALUES(?,?,?,?)");
EXEC SQL PREPARE DEPT_INSERT FROM :s;
  .
  .
(Check for successful execution and put values into :h1, :h2, :h3, :h4)
  .
  .
EXEC SQL EXECUTE DEPT_INSERT USING :h1, :h2,
:h3, :h4;
```

Example 2: This EXECUTE statement uses an SQLDA.

```
EXECUTE S3 USING DESCRIPTOR :sqlda3
```

Example 3: Given a procedure to award an employee a bonus:

```
CREATE PROCEDURE GIVE_BONUS (IN EMPNO INTEGER,
                             IN DEPTNO INTEGER,
                             OUT CHEQUE INTEGER,
                             INOUT BONUS DEC(6,0))
...
```

Dynamically call the procedure from a C application. The procedure takes the following host variables as input:

- *employee*, the ID number of the employee
- *dept*, the department number
- *bonus*, the bonus to be awarded to the employee

The procedure returns the following values to the host variables:

- *cheque_no*, the ID number from the cheque
- *bonus*, the actual bonus amount (after any adjustments)

```
strcpy (s, "CALL GIVE_BONUS(?, ?, ?, ?)");
EXEC SQL PREPARE DO_BONUS FROM :s;
  .
  .
/* Check for successful execution and put values into
   :employee, :dept, and :bonus */
  .
  .
EXEC SQL EXECUTE DO_BONUS INTO :cheque_no, :bonus
                          USING :employee, :dept, :bonus;
  .
  .
/* Check for successful execution and process the
   values returned in :cheque_no and :bonus */
```

EXECUTE IMMEDIATE

The EXECUTE IMMEDIATE statement prepares an executable form of an SQL statement from a character string form of the statement, and executes the SQL statement.

EXECUTE IMMEDIATE combines the basic functions of the PREPARE and EXECUTE statements. It can be used to prepare and execute SQL statements that contain neither host variables nor parameter markers.

Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

The authorization rules are those defined for the specified SQL statement.

The authorization ID of the statement might be affected by the DYNAMICRULES bind option.

Syntax

Description

expression
An expression returning the statement string to be executed. The expression must return a character-string type that is less than the maximum statement size of 2 097 152 bytes. Note that a CLOB(2097152) can contain a maximum size statement, but a VARCHAR cannot.

The statement string must be one of the following SQL statements:

- ALTER
- CALL
- COMMENT
- COMMIT
- Compound SQL (compiled)
- Compound SQL (inlined)
- CREATE
- DECLARE GLOBAL TEMPORARY TABLE
- DELETE
- DROP
- EXPLAIN
- FLUSH EVENT MONITOR
- FLUSH PACKAGE CACHE
- GRANT
- INSERT
- LOCK TABLE
- MERGE

- REFRESH TABLE
- RELEASE SAVEPOINT
- RENAME
- REVOKE
- ROLLBACK
- SAVEPOINT
- SET COMPILATION ENVIRONMENT
- SET CURRENT DECFLOAT ROUNDING MODE
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT FEDERATED ASYNCHRONY
- SET CURRENT IMPLICIT XMLPARSE OPTION
- SET CURRENT ISOLATION
- SET CURRENT LOCALE LC_MESSAGES
- SET CURRENT LOCALE LC_TIME
- SET CURRENT LOCK TIMEOUT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT MDC ROLLOUT MODE
- SET CURRENT OPTIMIZATION PROFILE
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET CURRENT TEMPORAL BUSINESS_TIME
- SET CURRENT TEMPORAL SYSTEM_TIME
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE (only if DYNAMICRULES run behavior is in effect for the package)
- SET INTEGRITY
- SET PASSTHRU
- SET PATH
- SET ROLE (only if DYNAMICRULES run behavior is in effect for the package)
- SET SCHEMA
- SET SERVER OPTION
- SET SESSION AUTHORIZATION
- SET SQL_CCFLAGS
- SET USAGE LIST STATE (only if DYNAMICRULES run behavior is in effect for the package)
- SET variable
- TRANSFER OWNERSHIP (only if DYNAMICRULES run behavior is in effect for the package)
- TRUNCATE (only if DYNAMICRULES run behavior is in effect for the package)
- UPDATE

The statement string must not include parameter markers or references to host variables, and must not begin with EXEC SQL. It must not contain a statement terminator, with the exception of compound SQL statements which can contain semi-colons (;) to separate statements within the compound block. A compound SQL statement is used within some CREATE and ALTER statements which, therefore, can also contain semi-colons.

When an EXECUTE IMMEDIATE statement is executed, the specified statement string is parsed and checked for errors. If the SQL statement is invalid, it is not executed, and the error condition that prevents its execution is reported in the SQLCA. If the SQL statement is valid, but an error occurs during its execution, that error condition is reported in the SQLCA.

Notes

- Statement caching affects the behavior of an EXECUTE IMMEDIATE statement.

Example

Use C program statements to move an SQL statement to the host variable *qstring* (char[80]), and prepare and execute whatever SQL statement is in the host variable qstring.

```
if ( strcmp(accounts,"BIG") == 0 )
  strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
    FROM EMP_ACT WHERE ACTNO < 100");
else
  strcpy (qstring,"INSERT INTO WORK_TABLE SELECT *
    FROM EMP_ACT WHERE ACTNO >= 100");
  .
  .
  .
EXEC SQL  EXECUTE IMMEDIATE :qstring;
```

EXPLAIN

The EXPLAIN statement captures information about the access plan chosen for the supplied explainable statement and places this information into the explain tables.

An *explainable statement* can either be a valid XQuery statement or one of the following SQL statements: CALL, Compound SQL (Dynamic), DELETE, INSERT, MERGE, REFRESH, SELECT, SELECT INTO, SET INTEGRITY, UPDATE, VALUES, or VALUES INTO.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

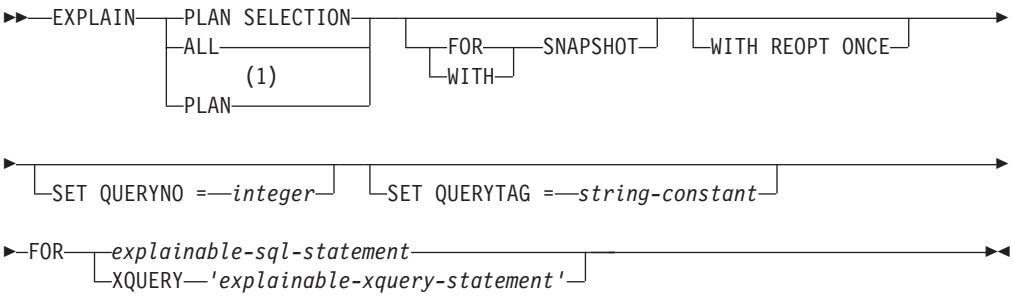
The statement to be explained is not executed.

Authorization

The authorization ID of the statement must hold at least one of the following authorizations:

- DATAACCESS authority which allows an INSERT, UPDATE, DELETE, or SELECT statement.
- INSERT privilege on the explain tables and at least one of the following authorizations:
  - All the privileges that are necessary to execute the explainable statement that is specified in the EXPLAIN statement (for example, if a DELETE statement is used as the explainable statement, the authorization rules for the DELETE statement are applied when the DELETE statement is explained)
  - EXPLAIN authority
  - SQLADM authority
  - DBADM authority

Syntax



Notes:

1 The PLAN option is supported only for syntax toleration of existing DB2 for z/OS EXPLAIN statements. There is no PLAN table. Specifying PLAN is equivalent to specifying PLAN SELECTION.

Description

PLAN SELECTION
Indicates that the information from the plan selection phase of query compilation is to be inserted into the explain tables.

ALL
Specifying ALL is equivalent to specifying PLAN SELECTION.

PLAN
The PLAN option provides syntax toleration for existing database applications from other systems. Specifying PLAN is equivalent to specifying PLAN SELECTION.

FOR SNAPSHOT
This clause indicates that only an explain snapshot is to be taken and placed into the SNAPSHOT column of the EXPLAIN_STATEMENT table. No other explain information is captured other than that present in the EXPLAIN_INSTANCE and EXPLAIN_STATEMENT tables.

WITH SNAPSHOT
This clause indicates that, in addition to the regular explain information, an explain snapshot is to be taken.

The default behavior of the EXPLAIN statement is to only gather regular explain information and not the explain snapshot.

default (neither FOR SNAPSHOT nor WITH SNAPSHOT specified)
Puts explain information into the explain tables.

WITH REOPT ONCE
This clause indicates that the specified *explainable statement* is to be reoptimized using the values for host variables, parameter markers, special registers, or global variables that were previously used to reoptimize this statement with REOPT ONCE. The explain tables will be populated with the new access plan. If the user has DBADM authority, or the database registry variable DB2_VIEW_REOPT_VALUES is set to YES, the EXPLAIN_PREDICATE table will also be populated with the values if they are used to reoptimize the statement.

SET QUERYNO = *integer*
Associates *integer*, via the QUERYNO column in the EXPLAIN_STATEMENT table, with the *explainable statement*. The integer value supplied must be a positive value.

If this clause is not specified for a dynamic EXPLAIN statement, a default value of one (1) is assigned. For a static EXPLAIN statement, the default value assigned is the statement number assigned by the precompiler.

SET QUERYTAG = *string-constant*
Associates *string-constant*, via the QUERYTAG column in the EXPLAIN_STATEMENT table, with the *explainable statement*. *string-constant* can be any character string up to 20 bytes in length. If the value supplied is less than 20 bytes in length, the value is padded on the right with blanks to the required length.

If this clause is not specified for an EXPLAIN statement, blanks are used as the default value.

FOR *explainable-sql-statement*
Specifies the SQL statement to be explained. This statement can be any valid CALL, Compound SQL (Dynamic), DELETE, INSERT, MERGE, REFRESH,

SELECT, SELECT INTO, SET INTEGRITY, UPDATE, VALUES, or VALUES INTO SQL statement. If the EXPLAIN statement is embedded in a program, the *explainable-sql-statement* can contain references to host variables (these variables must be defined in the program). Similarly, if EXPLAIN is being dynamically prepared, the *explainable-sql-statement* can contain parameter markers.

The *explainable-sql-statement* must be a valid SQL statement that could be prepared and executed independently of the EXPLAIN statement. It cannot be a statement name or host variable. SQL statements referring to cursors defined through CLP are not valid for use with this statement.

To explain dynamic SQL within an application, the entire EXPLAIN statement must be dynamically prepared.

FOR XQUERY *'explainable-xquery-statement'*
Specifies the XQUERY statement to be explained. This statement can be any valid XQUERY statement.

If the EXPLAIN statement is embedded in a program, the *'explainable-xquery-statement'* can contain references to host variables, provided that the host variables are not used in the top level XQUERY statement, but are passed in through an XMLQUERY function, by an XMLEXISTS predicate, or by an XMLTABLE function. The host variables must be defined in the program.

Similarly, if EXPLAIN is being dynamically prepared, the *'explainable-xquery-statement'* can contain parameter markers, provided that the same restrictions as for passing host variables are followed.

Alternatively, the DB2 XQUERY function db2-fn:sqlquery can be used to embed SQL statements with references to host variables and parameter markers.

The *'explainable-xquery-statement'* must be a valid XQUERY statement that could be prepared and executed independently of the EXPLAIN statement. Query statements referring to cursors defined through CLP are not valid for use with this statement.

Notes

- The Explain facility uses the following IDs as the schema when qualifying explain tables that it is populating:
  - The session authorization ID for dynamic SQL
  - The statement authorization ID for static SQL

  The schema can be associated with a set of explain tables, or aliases that point to a set of explain tables under a different schema. If no explain tables are found under the schema, the Explain facility checks for explain tables under the SYSTOOLS schema and attempts to use those tables.
- The following table shows the interaction of the snapshot keywords and the explain information.

Keyword Specified	Capture Explain Information?
none	Yes
FOR SNAPSHOT	No
WITH SNAPSHOT	Yes

If neither the FOR SNAPSHOT nor the WITH SNAPSHOT clause is specified, an explain snapshot is not taken.

- The explain tables must be created by the user before invocation of the EXPLAIN statement. The information generated by this statement is stored in the explain tables, in the schema that is designated at the time the statement is compiled.
- If any errors occur during the compilation of the *explainable statement* supplied, then no information is stored in the explain tables.
- The access plan generated for the *explainable statement* is not saved and thus, cannot be invoked at a later time. The explain information for the *explainable statement* is inserted when the EXPLAIN statement itself is compiled.
- For a static EXPLAIN query statement, the information is inserted into the explain tables at bind time and during an explicit rebind. During precompilation, the static EXPLAIN statements are commented out in the modified application source file. At bind time, the EXPLAIN statements are stored in the SYSCAT.STATEMENTS catalog. When the package is run, the EXPLAIN statement is not executed. Note that the section numbers for all statements in the application will be sequential and will include the EXPLAIN statements. An alternative to using a static EXPLAIN statement is to use a combination of the EXPLAIN and EXPLSNAP BIND or PREP options. Static EXPLAIN statements can be used to cause the explain tables to be populated for one specific static query statement out of many; simply prefix the target statement with the appropriate EXPLAIN statement syntax and bind the application without using either of the explain BIND or PREP options. The EXPLAIN statement can also be used when it is advantageous to set the QUERYNO or QUERYTAG field at the time of the actual explain invocation.
- Static EXPLAIN statements in an SQL procedure are evaluated when the procedure is compiled.
- For an incremental bind EXPLAIN query statement, the explain tables are populated when the EXPLAIN statement is submitted for compilation. When the package is run, the EXPLAIN statement performs no processing (though the statement will be successful). When populating the explain tables, the explain table qualifier and authorization ID used during population will be those of the package owner. The EXPLAIN statement can also be used when it is advantageous to set the QUERYNO or QUERYTAG field at the time of the actual explain invocation.
- For dynamic EXPLAIN statements, the explain tables are populated at the time the EXPLAIN statement is submitted for compilation. An EXPLAIN statement can be prepared with the PREPARE statement but, if executed, will perform no processing (though the statement will be successful). An alternative to issuing dynamic EXPLAIN statements is to use a combination of the CURRENT EXPLAIN MODE and CURRENT EXPLAIN SNAPSHOT special registers to explain dynamic query statements. The EXPLAIN statement should be used when it is advantageous to set the QUERYNO or QUERYTAG field at the time of the actual EXPLAIN invocation.
- If the REOPT bind option is set to ONCE, and either the CURRENT EXPLAIN MODE or the CURRENT EXPLAIN SNAPSHOT special register is set to REOPT, the execution of static and dynamic query statements containing host variables, special registers, parameter markers, or global variables will cause explain information to be captured for the statement only when the statement is reoptimized. Alternatively, if the REOPT bind option is set to ALWAYS, explain information will be captured every time these statements are executed.

Examples

- *Example 1:* Explain a simple SELECT statement and tag with QUERYNO = 13.

```
EXPLAIN PLAN SET QUERYNO = 13
  FOR SELECT C1
  FROM T1
```

- *Example 2:* Explain a simple SELECT statement and tag with QUERYTAG = 'TEST13'.

```
EXPLAIN PLAN SELECTION SET QUERYTAG = 'TEST13'
  FOR SELECT C1
  FROM T1
```

- *Example 3:* Explain a simple SELECT statement and tag with QUERYNO = 13 and QUERYTAG = 'TEST13'.

```
EXPLAIN PLAN SELECTION SET QUERYNO = 13 SET QUERYTAG = 'TEST13'
  FOR SELECT C1
  FROM T1
```

- *Example 4:* Attempt to get explain information when explain tables do not exist.

```
EXPLAIN ALL FOR SELECT C1
  FROM T1
```

  This statement will fail because the explain tables have not been defined (SQLSTATE 42704).

- *Example 5:* The following statement will succeed if it is found in the package cache and has already been compiled using REOPT ONCE.

```
EXPLAIN ALL WITH REOPT ONCE FOR SELECT C1
  FROM T1
  WHERE C1 = :<host variable>
```

- *Example 6:* The following example uses the db2-fn:xmlcolumn function, which takes the case- sensitive name of an XML column as an argument and returns an XML sequence that is the concatenation of XML column values.

  Consider a table called BUSINESS.CUSTOMER with an XML column called INFO. A simple XQuery that returns all documents from the INFO column is :

```
EXPLAIN PLAN SELECTION
   FOR XQUERY 'db2-fn:xmlcolumn ("BUSINESS.CUSTOMER.INFO")'
```

  If a column value is null, then the resulting return sequence for that row will be empty.

FETCH

The FETCH statement positions a cursor on the next row of its result table and assigns the values of that row to target variables.

Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared. When invoked using the command line processor, the syntax following *cursor-name* is optional and different from the SQL syntax. For more information, refer to "Using command line SQL statements and XQuery statements".

Authorization

For each global variable used as a *cursor-variable-name* or in the expression for an *array-index*, the privileges held by the authorization ID of the statement must include one of the following:

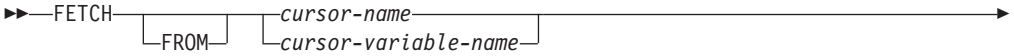
- READ privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

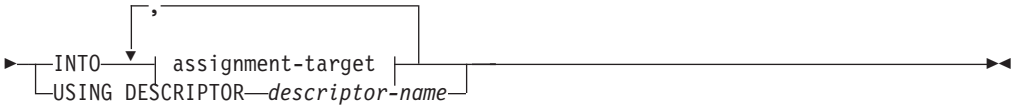
For each global variable used as an *assignment-target*, the privileges held by the authorization ID of the statement must include one of the following:

- WRITE privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

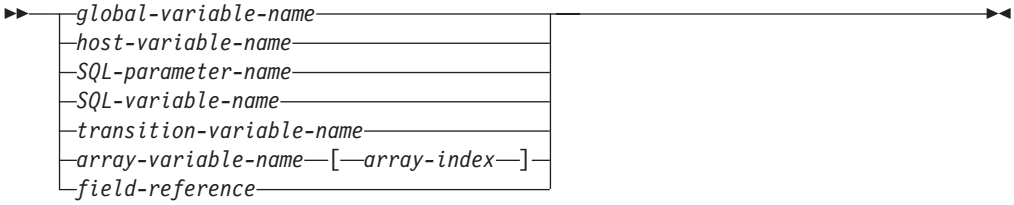
For the authorization required to use a cursor, see "DECLARE CURSOR".

Syntax





assignment-target



Description

cursor-variable-name
Identifies the cursor to be used in the fetch operation. The cursor-variable-name must identify a cursor variable that is in scope. When the FETCH statement is executed, the underlying cursor of the *cursor-variable-name* must be in the open state. A FETCH statement using a *cursor-variable-name* can only be used within a compound SQL (compiled) statement.

INTO *assignment-target*

Identifies one or more targets for the assignment of output values. The first value in the result row is assigned to the first target in the list, the second value to the second target, and so on. Each assignment to an *assignment-target* is made in sequence through the list. If an error occurs on any assignment, the value is not assigned to the target, and no more values are assigned to targets. Any values that have already been assigned to targets remain assigned.

When the data type of every *assignment-target* is not a row type, then the value 'W' is assigned to the SQLWARN3 field of the SQLCA if the number of *assignment-targets* is less than the number of result column values.

If the data type of an *assignment-target* is a row type, then there must be exactly one *assignment-target* specified (SQLSTATE 428HR), the number of columns must match the number of fields in the row type, and the data types of the columns of the fetched row must be assignable to the corresponding fields of the row type (SQLSTATE 42821).

If the data type of an *assignment-target* is an array element, then there must be exactly one *assignment-target* specified.

global-variable-name
Identifies the global variable that is the assignment target.

host-variable-name
Identifies the host variable that is the assignment target. For LOB output values, the target can be a regular host variable (if it is large enough), a LOB locator variable, or a LOB file reference variable.

SQL-parameter-name
Identifies the parameter that is the assignment target.

SQL-variable-name
Identifies the SQL variable that is the assignment target. SQL variables must be declared before they are used.

transition-variable-name
Identifies the column to be updated in the transition row. A *transition-variable-name* must identify a column in the subject table of a trigger, optionally qualified by a correlation name that identifies the new value.

array-variable-name
Identifies an SQL variable, SQL parameter, or global variable of an array type.

[array-index]
An expression that specifies which element in the array will be the target of the assignment. For an ordinary array, the *array-index* expression must be assignable to INTEGER (SQLSTATE 428H1) and cannot be the null value. Its value must be between 1 and the maximum cardinality defined for the array (SQLSTATE 2202E). For an

associative array, the *array-index* expression must be assignable to the index data type of the associative array (SQLSTATE 428H1) and cannot be the null value.

field-reference
Identifies the field within a row type value that is the assignment target. The *field-reference* must be specified as a qualified *field-name* where the qualifier identifies the row value in which the field is defined.

USING DESCRIPTOR *descriptor-name*
Identifies an SQLDA that must contain a valid description of zero or more host variables.

Before the FETCH statement is processed, the user must set the following fields in the SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA.
- SQLDABC to indicate the number of bytes of storage allocated for the SQLDA.
- SQLD to indicate the number of variables used in the SQLDA when processing the statement.
- SQLVAR occurrences to indicate the attributes of the variables.

The SQLDA must have enough storage to contain all SQLVAR occurrences. Therefore, the value in SQLDABC must be greater than or equal to 16 + SQLN*(N), where N is the length of an SQLVAR occurrence.

If LOB or structured type result columns need to be accommodated, there must be two SQLVAR entries for every select-list item (or column of the result table).

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN.

The *n*th variable described in the SQLDA corresponds to the *n*th column of the result table of the cursor. The data type of each variable must be compatible with its corresponding column.

Each assignment to a variable is made according to specific rules. If the number of variables is less than the number of values in the row, the SQLWARN3 field of the SQLDA is set to 'W'. Note that there is no warning if there are more variables than the number of result columns. If an assignment error occurs, the value is not assigned to the variable, and no more values are assigned to variables. Any values that have already been assigned to variables remain assigned.

Notes

- ***Cursor position:*** An open cursor has three possible positions:
  - Before a row
  - On a row
  - After the last row.

  A cursor can only be on a row as a result of a FETCH statement. If the cursor is currently positioned on or after the last row of the result table:
  - SQLCODE is set to +100, and SQLSTATE is set to '02000'.
  - The cursor is positioned after the last row.
  - Values are not assigned to assignment targets.

If the cursor is currently positioned before a row, it will be repositioned on that row, and values will be assigned to targets as specified by the INTO or USING clause.

If the cursor is currently positioned on a row other than the last row, it will be repositioned on the next row and values of that row will be assigned to targets as specified by the INTO or USING clause.

If a cursor is on a row, that row is called the current row of the cursor. A cursor referenced in an UPDATE or DELETE statement must be positioned on a row.

It is possible for an error to occur that makes the state of the cursor unpredictable.

- When retrieving into LOB locators in situations where it is not necessary to retain the locator across FETCH statements, it is good practice to issue a FREE LOCATOR statement before issuing the next FETCH statement, as locator resources are limited.
- It is possible that a warning may not be returned on a FETCH. It is also possible that the returned warning applies to a previously fetched row. This occurs as a result of optimizations such as the use of system temporary tables or pushdown operators.
- Statement caching affects the behavior of an EXECUTE IMMEDIATE statement.
- DB2 CLI supports additional fetching capabilities. For instance when a cursor's result table is read-only, the SQLFetchScroll() function can be used to position the cursor at any spot within that result table.
- For an updatable cursor, a lock is obtained on a row when it is fetched.
- If the cursor definition contains an SQL data change statement or invokes a routine that modifies SQL data, an error during the fetch operation does not cause the modified rows to be rolled back, even if the error results in the cursor being closed.

Examples

- *Example 1:* In this C example, the FETCH statement fetches the results of the SELECT statement into the program variables `dnum`, `dname`, and `mnum`. When no more rows remain to be fetched, the not found condition is returned.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO FROM TDEPT
    WHERE ADMRDEPT = 'A00';

EXEC SQL OPEN C1;

while (SQLCODE==0) {
    EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;
}

EXEC SQL CLOSE C1;
```

- *Example 2:* This FETCH statement uses an SQLDA.

```
FETCH CURS USING DESCRIPTOR :sqlda3
```

FLUSH BUFFERPOOLS

The FLUSH BUFFERPOOLS statement writes the dirty pages from all the local buffer pools for a particular database synchronously to disk. In DB2 pureScale environments, the dirty pages in the group buffer pool are also written synchronously to disk.

This statement is not under transaction control.

The FLUSH BUFFERPOOLS statement can be used in the following ways:

- To reduce the recovery window of a database in the event of a failure
- To reduce the size of logs written to a backup image before database operations such as online backups
- To minimize the recovery time of a split-mirror database

Invocation

The statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include SQLADM, DBADM, SYSMAINT, SYSCTRL, or SYSADM authority.

Syntax

Description

ALL
Flushes the dirty pages from all the buffer pools (local and group).

Notes

- **Dirty pages processing:** Only the dirty pages that are in the buffer pools when the statement begins processing are written to disk. Any dirty pages that are added to the buffer pools before the statement finishes processing are not written to disk.
- **Syntax alternatives:** BUFFERPOOL can be specified in place of BUFFERPOOLS.

FLUSH EVENT MONITOR

The FLUSH EVENT MONITOR statement writes current database monitor values for all active monitor types associated with event monitor *event-monitor-name* to the event monitor I/O target.

A partial event record is available at any time for event monitors that have low record generation frequency (such as a database event monitor). Such records are noted in the event monitor log with a *partial record* identifier.

When an event monitor is flushed, its active internal buffers are written to the event monitor output object.

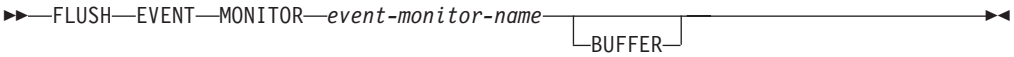
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include SQLADM or DBADM authority.

Syntax



Description

event-monitor-name
Name of the event monitor. This is a one-part name. It is an ordinary identifier.

BUFFER
Indicates that the event monitor buffers are to be written out. If BUFFER is specified, then a partial record is not generated. Only the data already present in the event monitor buffers are written out.

Notes

- Flushing out the event monitor will not cause the event monitor values to be reset. This means that the event monitor record that would have been generated if no flush was performed, will still be generated when the normal monitor event is triggered.
- The FLUSH EVENT MONITOR statement does not cause events to be generated and written for the UNIT OF WORK event monitor.

FLUSH FEDERATED CACHE

The FLUSH FEDERATED CACHE statement flushes the federated cache, allowing fresh metadata to be obtained the next time an SQL statement is issued against the remote table or view using a federated three part name.

When an SQL statement is issued against a remote table or view using a federated three part name, if the remote table or view is being referenced for the first time, the metadata and statistics for the remote object are retrieved and stored in a federated cache.

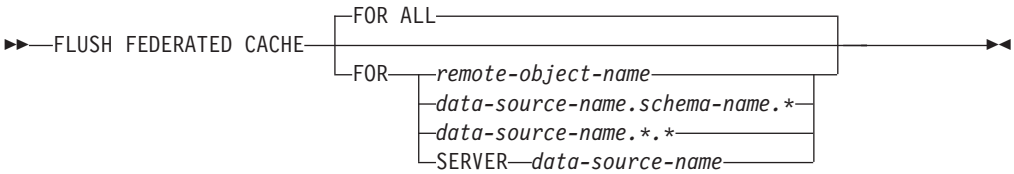
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include either SQLADM or DBADM authority.

Syntax



Description

FOR ALL
Flushes the federated cache information for all objects from all data sources. This is the default.

FOR *remote-object-name*
Flushes the federated cache information for a specific remote table or view.

FOR *data-source-name.schema-name.**
Flushes the federated cache information for all objects in the schema identified by *schema-name* from the specific data source identified by *data-source-name*.

FOR *data-source-name.*.**
Flushes the federated cache information for all objects from the specific data source identified by *data-source-name*.

FOR SERVER *data-source-name*
Flushes the federated cache information for all objects from the specific data source identified by *data-source-name*.

Notes

- **Package invalidation:** Flushing the federated cache causes packages with a dependency on the three-part name to be invalidated. This action could have a performance impact since the invalidated packages need to be recompiled whenever statements from the package are executed.

- **View invalidation:** Flushing the federated cache will not cause the views depending on the three part name to be invalidated. The next time the view is used, it will implicitly revalidate the view. If there are changes to the remote object, it is possible that the statement using the view could return an error.

Examples

- *Example 1*: Flush the federated cache information for the remote-table-name *t1* in the remote-schema-name *rschema* on the data source *rudb*.

  ```
  FLUSH FEDERATED CACHE FOR rudb.rschema.t1
  ```

- *Example 2*: Flush the federated cache information for all objects in the remote-schema-name *rschema* on the data source *rudb*.

  ```
  FLUSH FEDERATED CACHE FOR rudb.rschema.*
  ```

- *Example 3*: Flush the federated cache information for all objects from the data source *rudb*.

  ```
  FLUSH FEDERATED CACHE FOR rudb.*.*
  ```

  An alternative to this syntax is as follows:

  ```
  FLUSH FEDERATED CACHE FOR SERVER rudb
  ```

FLUSH OPTIMIZATION PROFILE CACHE

Multiple statements can be compiled using the same optimization profile. To make optimization profile processing more efficient, the optimization profile is processed the first time it is used to optimize a statement, and the output is stored in the optimization profile cache. Subsequent references to the optimization profile use the processed version in the optimization profile cache.

An optimization profile should be removed from the optimization profile cache when the version stored in SYSTOOLS.OPT_PROFILE has been updated. When the old version is removed from the cache, the new version will be used upon optimization of subsequent statements that use the optimization profile.

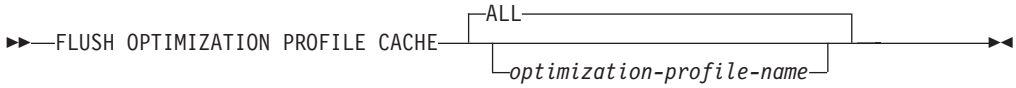
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include either SQLADM or DBADM authority (SQLSTATE 42502).

Syntax



Description

optimization-profile-name
Specifies the name of the optimization profile to be flushed from the optimization profile cache. If the name specified is unqualified, the value of the CURRENT DEFAULT SCHEMA register is used as the implicit qualifier.

ALL
Specifies that all profiles on all active database partitions be flushed from the optimization profile cache.

Notes

- The FLUSH OPTIMIZATION PROFILE CACHE statement removes all or a single optimization profile from the optimization profile cache. It also causes the logical invalidation of any cached dynamic SQL statements that were prepared with that optimization profile.
- New access plans for any invalidated dynamic plans are regenerated when the next request for the same SQL statement is made.
- Packages that reference an optimization profile removed from the optimization profile cache by this statement must be explicitly bound again to allow new access plans to be generated.

Examples

- *Example 1:* The optimization profile "Rick"."Foo" is flushed from the optimization profile cache.

```
SET CURRENT SCHEMA = '"Rick"'
FLUSH OPTIMIZATION PROFILE CACHE "Foo"
```

- *Example 2:* The optimization profile JOHN.ALL is removed from the optimization profile cache.

```
SET CURRENT SCHEMA = '"Rick"'
FLUSH OPTIMIZATION PROFILE CACHE JOHN.ALL
```

Messages

- No errors are issued if the optimization profile cache is empty or if the specified optimization profiles (specified explicitly or implicitly) do not exist in the optimization profile cache.

FLUSH PACKAGE CACHE

The FLUSH PACKAGE CACHE statement removes all cached dynamic SQL statements currently in the package cache. This statement causes the logical invalidation of any cached dynamic SQL statement and forces the next request for the same SQL statement to be implicitly compiled by DB2.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include SQLADM or DBADM authority.

Syntax

Notes

- This statement affects all cached dynamic SQL entries in the package cache on all active database partitions.
- As cached dynamic SQL statements are invalidated, the package cache memory used for the cached entry will be freed if the entry is not in use when the FLUSH PACKAGE CACHE statement executes.
- Any cached dynamic SQL statement currently in use will be allowed to continue to exist in the package cache until it is no longer needed by the its current user; the next new user of the same statement will force an implicit prepare of the statement by DB2, and the new user will execute the new version of the cached dynamic SQL statement.

FOR

The FOR statement executes a statement or group of statements for each row of a table.

Invocation

This statement can be embedded in an:

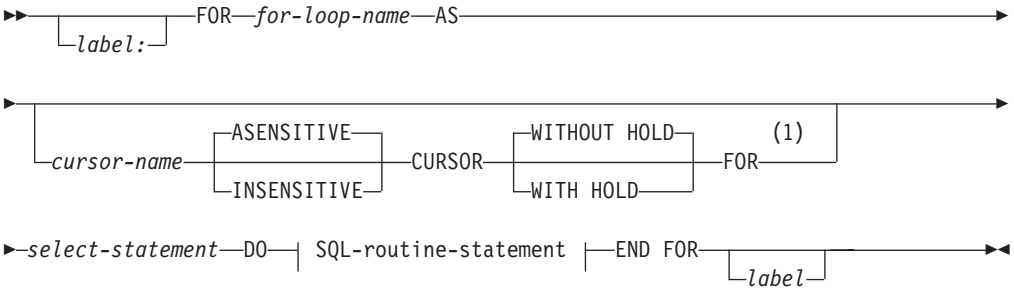
- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

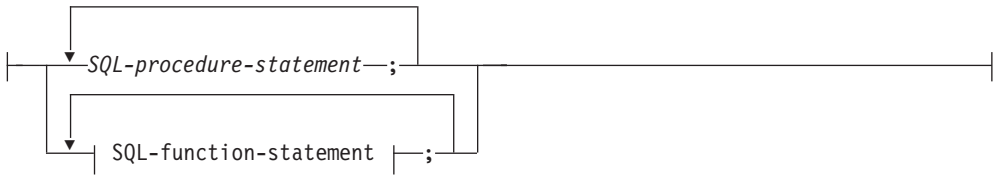
Authorization

No privileges are required to invoke the FOR statement. However, the authorization ID of the statement must hold the necessary privileges to invoke the SQL statements that are embedded in the FOR statement. For the authorization required to use a cursor, see "DECLARE CURSOR".

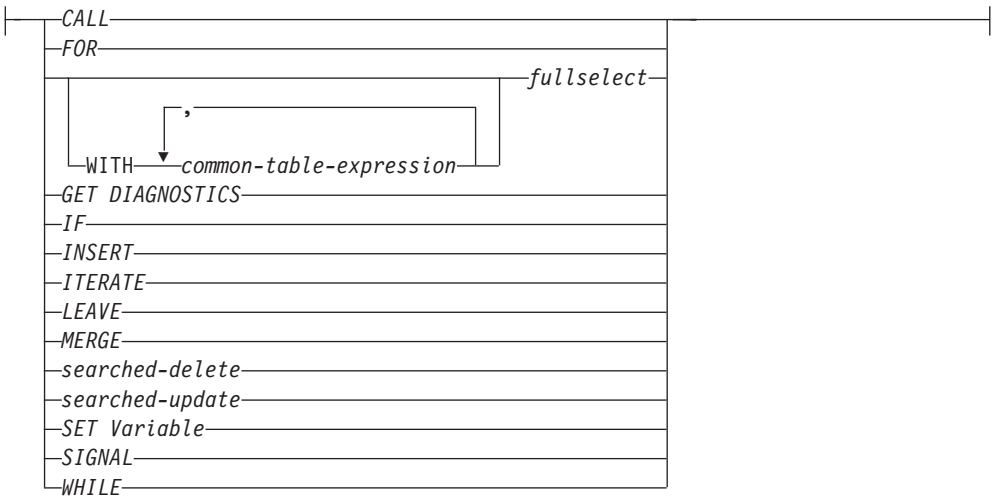
Syntax



SQL-routine-statement:



SQL-function-statement:



Notes:

1 This option can only be used in the context of an SQL procedure or a compound SQL (compiled) statement.

Description

label
Specifies the label for the FOR statement. If the beginning label is specified, that label can be used in LEAVE and ITERATE statements. If the ending label is specified, it must be the same as the beginning label.

for-loop-name
Specifies a label for the implicit compound statement generated to implement the FOR statement. It follows the rules for the label of a compound statement except that it cannot be used with an ITERATE or LEAVE statement within the FOR statement. The *for-loop-name* is used to qualify the column names returned by the specified *select-statement*.

cursor-name
Names the cursor that is used to select rows from the result table of the SELECT statement. If not specified, DB2 generates a unique cursor name. For a description of ASENSITIVE, INSENSITIVE, WITHOUT HOLD, or WITH HOLD, see "DECLARE CURSOR".

select-statement
Specifies the SELECT statement of the cursor. All columns in the select list must have a name and there cannot be two columns with the same name.

In a trigger, function, method, or compound SQL (inlined) statement, the *select-statement* must consist of only a *fullselect* with optional common table expressions.

SQL-procedure-statement
Specifies one or more statements to be invoked for each row of the table. *SQL-procedure-statement* is only applicable when in the context of an SQL procedure or within a compound SQL (compiled) statement. See *SQL-procedure-statement* in "Compound SQL (compiled)" statement.

SQL-function-statement
Specifies one or more statements to be invoked for each row of the table. A

searched-update, searched-delete, or INSERT operation on nicknames is not supported. *SQL-function-statement* is only applicable when in the context of an SQL function or SQL method.

Rules

- The select list must consist of unique column names and the objects specified in the *select-statement* must exist when the procedure is created, or the object must be created in a previous SQL procedure statement.
- The cursor specified in a for-statement cannot be referenced outside the for-statement and cannot be specified in an OPEN, FETCH, or CLOSE statement.

Example

In the following example, the for-statement is used to iterate over the entire `employee` table. For each row in the table, the SQL variable `fullname` is set to the last name of the employee, followed by a comma, the first name, a blank space, and the middle initial. Each value for `fullname` is inserted into table `tnames`.

```
BEGIN ATOMIC
  DECLARE fullname CHAR(40);
  FOR vl AS
    SELECT firstnme, midinit, lastname FROM employee
      DO
      SET fullname = lastname CONCAT ','
        CONCAT firstnme CONCAT ' ' CONCAT midinit;
      INSERT INTO tnames VALUES (fullname);
  END FOR;
END
```

FREE LOCATOR

The FREE LOCATOR statement removes the association between a locator variable and its value.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax

Description

LOCATOR *variable-name*, **...**

Identifies one or more locator variables that must be declared in accordance with the rules for declaring locator variables.

The locator-variable must currently have a locator assigned to it. That is, a locator must have been assigned during this unit of work (by a CALL, FETCH, SELECT INTO, or VALUES INTO statement) and must not subsequently have been freed (by a FREE LOCATOR statement); otherwise, an error is returned (SQLSTATE 0F001).

If more than one locator is specified, all locators that can be freed will be freed, regardless of errors detected in other locators in the list.

Example

In a COBOL program, free the BLOB locator variables TKN-VIDEO and TKN-BUF and the CLOB locator variable LIFE-STORY-LOCATOR.

```
EXEC SQL
FREE LOCATOR :TKN-VIDEO, :TKN-BUF, :LIFE-STORY-LOCATOR
END-EXEC.
```

GET DIAGNOSTICS

The GET DIAGNOSTICS statement is used to obtain current execution environment information including information about the previous SQL statement (other than a GET DIAGNOSTICS statement) that was executed. Some of the information available through the GET DIAGNOSTICS statement is also available in the SQLCA.

Invocation

This statement can be embedded in an:

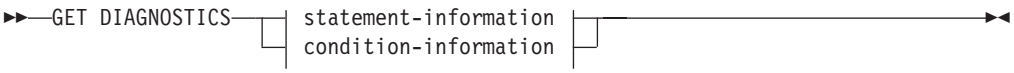
- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

Authorization

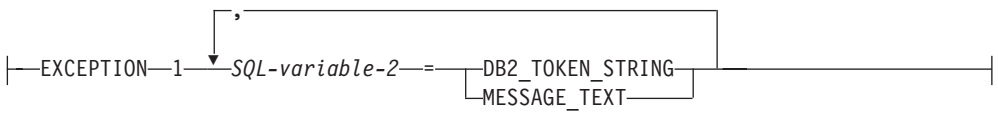
None required.

Syntax



statement-information:

condition-information:



Description

statement-information
Returns information about the last SQL statement executed.

SQL-variable-1
Identifies the variable that is the assignment target. The variable must not be a global variable. SQL variables can be defined in a compound statement. The data type of the variable must be compatible with the data type as specified in Table 33 on page 1036.

DB2_RETURN_STATUS
Identifies the status value returned from the procedure associated with the previously executed SQL statement, provided that the statement was a

CALL statement invoking a procedure that returns a status. If the previous statement is not such a statement, then the value returned has no meaning and could be any integer.

DB2_SQL_NESTING_LEVEL
Identifies the current level of nesting or recursion in effect when the GET DIAGNOSTICS statement was executed. Each level of nesting corresponds to a nested or recursive invocation of a compiled SQL function, compiled SQL procedure, compiled trigger, or dynamically prepared compound SQL (compiled) statement. If the GET DIAGNOSTICS statement is executed outside of a level of nesting, the value zero is returned. This option can be specified only in the context of a compiled SQL function, compiled SQL procedure, compiled trigger, or compound SQL (compiled) statement (SQLSTATE 42601).

ROW_COUNT
Identifies the number of rows associated with the previous SQL statement. If the previous SQL statement is a DELETE, INSERT, or UPDATE statement, ROW_COUNT identifies the number of rows that qualified for the operation. If the previous statement is a PREPARE statement, ROW_COUNT identifies the *estimated* number of result rows in the prepared statement.

condition-information
Specifies that the error or warning information for the previously executed SQL statement is to be returned. If information about an error is needed, the GET DIAGNOSTICS statement must be the first statement specified in the handler that will handle the error. If information about a warning is needed, and if the handler will get control of the warning condition, the GET DIAGNOSTICS statement must be the first statement specified in that handler. If the handler will *not* get control of the warning condition, the GET DIAGNOSTICS statement must be the next statement executed. This option can only be specified in the context of an SQL Procedure (SQLSTATE 42601).

SQL-variable-2
Identifies the variable that is the assignment target. The variable must not be a global variable. SQL variables can be defined in a compound statement. The data type of the variable must be compatible with the data type as specified in Table 33 on page 1036.

DB2_TOKEN_STRING
Identifies any error or warning message tokens returned from the previously executed SQL statement. If the statement completed with an SQLCODE of zero, or if the SQLCODE had no tokens, an empty string is returned for a VARCHAR variable or blanks are returned for a CHAR variable.

MESSAGE_TEXT
Identifies any error or warning message text returned from the previously executed SQL statement. The message text is returned in the language of the database server where the statement is processed. If the statement completed with an SQLCODE of zero, an empty string is returned for a VARCHAR variable or blanks are returned for a CHAR variable.

Notes

- The GET DIAGNOSTICS statement does not change the contents of the diagnostics area (SQLCA). If an SQLSTATE or SQLCODE special variable is declared in the SQL procedure, these are set to the SQLSTATE or SQLCODE returned from issuing the GET DIAGNOSTICS statement.

- **Data types for items**: The following table shows the SQL data type for each diagnostic item. When a diagnostic item is assigned to a variable, the data type of the variable must be compatible with the data type of the requested diagnostic item.

Table 33. Data types for GET DIAGNOSTICS items

Type of information	Item	Data type
Statement information	DB2_RETURN_STATUS	INTEGER
Statement information	DB2_SQL_NESTING_LEVEL	INTEGER
Statement information	ROW_COUNT	DECIMAL(31,0)
Condition information	DB2_TOKEN_STRING	VARCHAR(1000)
Condition information	MESSAGE_TEXT	VARCHAR(32672)

- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  - RETURN_STATUS can be specified in place of DB2_RETURN_STATUS.

Examples

- *Example 1*: In an SQL procedure, execute a GET DIAGNOSTICS statement to determine how many rows were updated.

```
CREATE PROCEDURE sqlprocg (IN deptnbr VARCHAR(3))
   LANGUAGE SQL
   BEGIN
      DECLARE SQLSTATE CHAR(5);
      DECLARE rcount INTEGER;
      UPDATE CORPDATA.PROJECT
         SET PRSTAFF = PRSTAFF + 1.5
         WHERE DEPTNO = deptnbr;
      GET DIAGNOSTICS rcount = ROW_COUNT;
-- At this point, rcount contains the number of rows that were updated.
...
   END
```

- *Example 2*: Within an SQL procedure, handle the returned status value from the invocation of a procedure called TRYIT that could either explicitly RETURN a positive value indicating a user failure, or encounter SQL errors that would result in a negative return status value. If the procedure is successful, it returns a value of zero.

```
CREATE PROCEDURE TESTIT ()
   LANGUAGE SQL
   A1:BEGIN
      DECLARE RETVAL INTEGER DEFAULT 0;
      ...
      CALL TRYIT;
      GET DIAGNOSTICS RETVAL = DB2_RETURN_STATUS;
      IF RETVAL <> 0 THEN
         ...
         LEAVE A1;
      ELSE
         ...
      END IF;
   END A1
```

GOTO

The GOTO statement is used to branch to a user-defined label within an SQL procedure.

Invocation

This statement can only be embedded in an SQL procedure. It is not an executable statement and cannot be dynamically prepared.

Authorization

None required.

Syntax

Description

label

Specifies a labelled statement where processing is to continue. The labelled statement and the GOTO statement must be in the same scope:

- If the GOTO statement is defined in a FOR statement, *label* must be defined inside the same FOR statement, excluding a nested FOR statement or nested compound statement
- If the GOTO statement is defined in a compound statement, *label* must be defined inside the same compound statement, excluding a nested FOR statement or nested compound statement
- If the GOTO statement is defined in a handler, *label* must be defined in the same handler, following the other scope rules
- If the GOTO statement is defined outside of a handler, *label* must not be defined within a handler.

If *label* is not defined within a scope that the GOTO statement can reach, an error is returned (SQLSTATE 42736).

Notes

- It is recommended that the GOTO statement be used sparingly. This statement interferes with normal processing sequences, thus making a routine more difficult to read and maintain. Before using a GOTO statement, determine whether another statement, such as IF or LEAVE, can be used in place, to eliminate the need for a GOTO statement.

Example

In the following compound statement, the parameters *rating* and *v_empno* are passed into the procedure, which then returns the output parameter *return_parm* as a date duration. If the employee's time in service with the company is less than 6 months, the GOTO statement transfers control to the end of the procedure, and *new_salary* is left unchanged.

```
CREATE PROCEDURE adjust_salary
  (IN v_empno CHAR(6),
  IN rating INTEGER,
  OUT return_parm DECIMAL (8,2))
```

```
  MODIFIES SQL DATA
  LANGUAGE SQL
  BEGIN
    DECLARE new_salary DECIMAL (9,2);
    DECLARE service DECIMAL (8,2);
      SELECT SALARY, CURRENT_DATE - HIREDATE
        INTO new_salary, service
        FROM EMPLOYEE
        WHERE EMPNO = v_empno;
      IF service < 600
        THEN GOTO EXIT;
      END IF;
      IF rating = 1
        THEN SET new_salary = new_salary + (new_salary * .10);
      ELSEIF rating = 2
        THEN SET new_salary = new_salary + (new_salary * .05);
      END IF;
      UPDATE EMPLOYEE
        SET SALARY = new_salary
        WHERE EMPNO = v_empno;
      EXIT: SET return_parm = service;
END
```

GRANT (database authorities)

This form of the GRANT statement grants authorities that apply to the entire database (rather than privileges that apply to specific objects within the database).

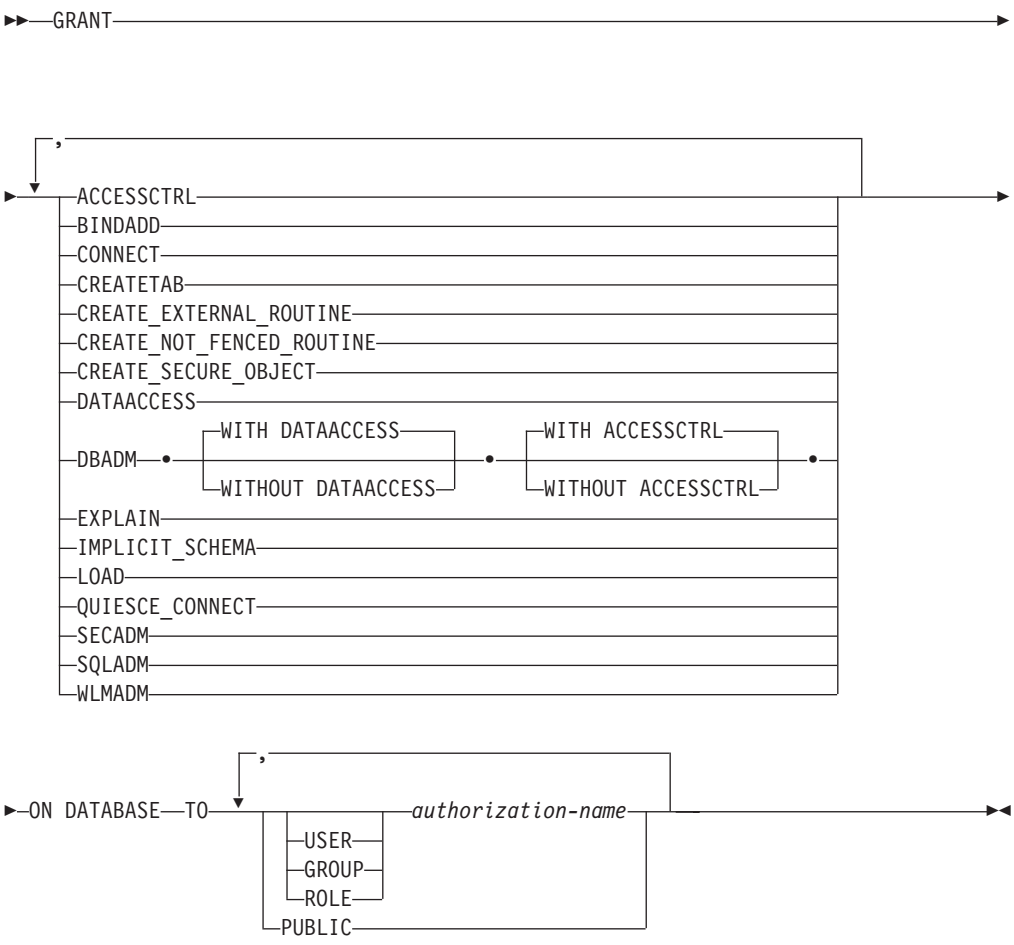
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

To grant ACCESSCTRL, CREATE_SECURE_OBJECT, DATAACCESS, DBADM, or SECADM authority, SECADM authority is required. To grant other authorities ACCESSCTRL or SECADM authority is required.

Syntax



Description

ACCESSCTRL

Grants the access control authority. The ACCESSCTRL authority allows the holder to:

- Grant and revoke the following database authorities: BINDADD, CONNECT, CREATETAB, CREATE_EXTERNAL_ROUTINE, CREATE_NOT_FENCED_ROUTINE, EXPLAIN, IMPLICIT_SCHEMA, LOAD, QUIESE_CONNECT, SQLADM, WLMADM
- Grant and revoke all object level privileges

The ACCESSCTRL authority cannot be granted to PUBLIC (SQLSTATE 42508).

BINDADD
Grants the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if the BINDADD authority is subsequently revoked.

CONNECT
Grants the authority to access the database.

CREATETAB
Grants the authority to create base tables. The creator of a base table automatically has the CONTROL privilege on that table. The creator retains this privilege even if the CREATETAB authority is subsequently revoked.

There is no explicit authority required for view creation. A view can be created at any time if the authorization ID of the statement used to create the view has either CONTROL or SELECT privilege on each base table of the view.

CREATE_EXTERNAL_ROUTINE
Grants the authority to register external routines. Care must be taken that routines so registered will not have adverse side effects. (For more information, see the description of the THREADSAFE clause on the CREATE or ALTER routine statements.)

Once an external routine has been registered, it continues to exist, even if CREATE_EXTERNAL_ROUTINE is subsequently revoked.

CREATE_NOT_FENCED_ROUTINE
Grants the authority to register routines that execute in the database manager's process. Care must be taken that routines so registered will not have adverse side effects. (For more information, see the description of the FENCED clause on the CREATE or ALTER routine statements.)

Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE_NOT_FENCED_ROUTINE is subsequently revoked.

CREATE_EXTERNAL_ROUTINE is automatically granted to an *authorization-name* that is granted CREATE_NOT_FENCED_ROUTINE authority.

CREATE_SECURE_OBJECT
Grants the authority to create secure triggers and secure functions. Grants the authority to alter the secure attribute of such objects as well.

DATAACCESS
Grants the authority to access data. The DATAACCESS authority allows the holder to:

- Select, insert, update, delete, and load data
- Execute any package
- Execute any routine (except audit routines)

The DATAACCESS authority cannot be granted to PUBLIC (SQLSTATE 42508).

DBADM
Grants the database administrator authority. A database administrator holds

nearly all privileges on nearly all objects in the database. The only exceptions are those privileges that are part of the access control, data access, and security administrator authorities. DBADM cannot be granted to PUBLIC.

EXPLAIN
Grants the authority to explain statements. The EXPLAIN authority allows the holder to explain, prepare, and describe dynamic and static SQL statements without requiring access to data.

IMPLICIT_SCHEMA
Grants the authority to implicitly create a schema.

LOAD
Grants the authority to load in this database. This authority gives a user the right to use the LOAD utility in this database. DATAACCESS and DBADM also have this authority by default. However, if a user only has LOAD authority (not DATAACCESS), the user is also required to have table-level privileges. In addition to LOAD privilege, the user is required to have:

- INSERT privilege on the table for LOAD with mode INSERT, TERMINATE (to terminate a previous LOAD INSERT), or RESTART (to restart a previous LOAD INSERT)
- INSERT and DELETE privilege on the table for LOAD with mode REPLACE, TERMINATE (to terminate a previous LOAD REPLACE), or RESTART (to restart a previous LOAD REPLACE)
- INSERT privilege on the exception table, if such a table is used as part of LOAD

QUIESCE_CONNECT
Grants the authority to access the database while it is quiesced.

SECADM
Grants the security administrator authority. The authority allows the holder to:

- Create and drop security objects such as audit policies, roles, security labels, security label components, security policies, and trusted contexts
- Grant and revoke authorities, exemptions, privileges, roles, and security labels
- Grant and revoke the SETSESSIONUSER privilege
- Execute TRANSFER OWNERSHIP on objects owned by others

The SECADM authority cannot be granted to PUBLIC (SQLSTATE 42508).

SQLADM
Grants the authority to manage SQL statement execution. The SQLADM authority allows the holder to:

- Create, drop, flush, and set event monitors
- Explain, prepare, and describe dynamic and static SQL statements without requiring access to data
- Flush optimization profile cache
- Flush package cache
- Execute the runstats utility
- Create, alter, drop, and set usage lists

WLMADM
Grants the authority to manage workloads. The WLMADM authority allows the holder to:

- Create, drop, and alter service classes, work action sets, work class sets, or workloads.

TO Specifies to whom the authorities are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the authorities to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

Notes

- ACCESSCTRL, CREATE_SECURE_OBJECT, DATAACCESS, DBADM, or SECADM authorities cannot be granted to the special group PUBLIC. Therefore, granting ACCESSCTRL, CREATE_SECURE_OBJECT, DBADM, DATAACCESS, or SECADM authority to a role *role-name* fails if *role-name* is granted to PUBLIC either directly or indirectly (SQLSTATE 42508).
  - Role *role-name* is granted directly to PUBLIC if the following statement has been issued:

    ```
    GRANT ROLE role-name TO PUBLIC
    ```

  - Role *role-name* is granted indirectly to PUBLIC if the following statements have been issued:

    ```
    GRANT ROLE role-name TO ROLE role-name2
    GRANT ROLE role-name2 TO PUBLIC
    ```

- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products.

  - CREATE_NOT_FENCED can be specified in place of CREATE_NOT_FENCED_ROUTINE
  - SYSTEM can be specified in place of DATABASE
- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Examples

- *Example 1:* Give the users WINKEN, BLINKEN, and NOD the authority to connect to the database.

```
GRANT CONNECT ON DATABASE TO USER WINKEN, USER BLINKEN, USER NOD
```

- *Example 2:* Grant BINDADD authority on the database to a group named D024. There is both a group and a user called D024 in the system.

```
GRANT BINDADD ON DATABASE TO GROUP D024
```

  Observe that, the GROUP keyword must be specified; otherwise, an error will occur since both a user and a group named D024 exist. Any member of the D024 group will be allowed to bind packages in the database, but the D024 user will not be allowed (unless this user is also a member of the group D024, had been granted BINDADD authority previously, or BINDADD authority had been granted to another group of which D024 was a member).

- *Example 3:* Give user Walid security administrator authority.

```
GRANT SECADM ON DATABASE TO USER Walid
```

- *Example 4:* A user with SECADM authority grants the CREATE_SECURE_OBJECT authority to user Haytham.

```
GRANT CREATE_SECURE_OBJECT ON DATABASE TO USER HAYTHAM
```

GRANT (exemption)

This form of the GRANT statement grants to a user, group, or role an exemption on an access rule for a specified label-based access control (LBAC) security policy.

When the user holding the exemption accesses data in a table protected by that security policy the indicated rule will not be enforced when deciding if they can access the data.

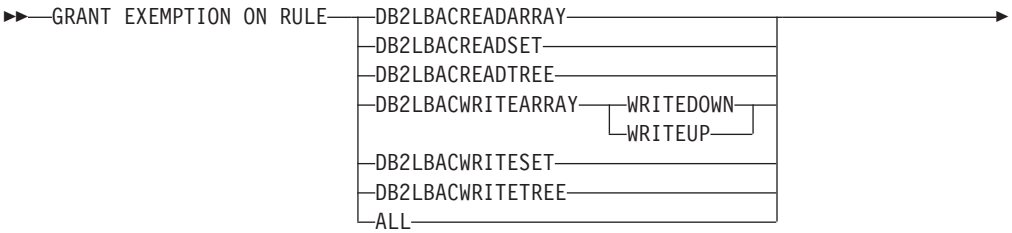
Invocation

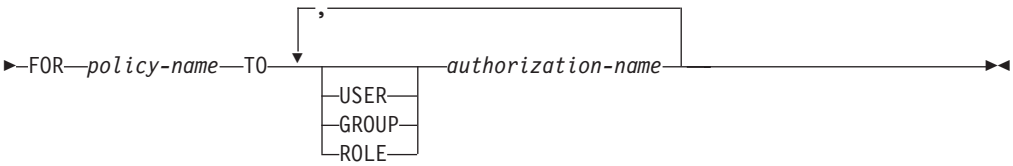
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax





Description

EXEMPTION ON RULE
Grants an exemption on an access rule.

DB2LBACREADARRAY
Grants an exemption on the predefined DB2LBACREADARRAY rule.

DB2LBACREADSET
Grants an exemption on the predefined DB2LBACREADSET rule.

DB2LBACREADTREE
Grants an exemption on the predefined DB2LBACREADTREE rule.

DB2LBACWRITEARRAY
Grants an exemption on the predefined DB2LBACWRITEARRAY rule.

WRITEDOWN
Specifies that the exemption only applies to write down.

WRITEUP
Specifies that the exemption only applies to write up.

DB2LBACWRITESET
Grants an exemption on the predefined DB2LBACWRITESET rule.

DB2LBACWRITETREE
Grants an exemption on the predefined DB2LBACWRITETREE rule.

ALL
Grants an exemption on all of the predefined rules.

FOR *policy-name*
Identifies the security policy for which the exemption is being granted. The exemption will only be effective for tables that are protected by this security policy. The name must identify a security policy already described in the catalog (SQLSTATE 42704).

TO Specifies to whom the exemption is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- If the security policy is not defined to consider access through groups or roles, any exemption granted to a group or role is ignored when access is attempted.

Notes

- By default when a security policy is created, only exemptions granted to an individual user are considered. To have groups or roles considered for the

security policy, you must issue the ALTER SECURITY POLICY statement and specify USE GROUP AUTHORIZATION or USE ROLE AUTHORIZATION as applicable.

Examples

- *Example 1:* Grant an exemption on access rule DB2LBACREADSET for security policy DATA_ACCESS to user WALID.

```
GRANT EXEMPTION ON RULE DB2LBACREADSET FOR DATA_ACCESS TO USER WALID
```

- *Example 2:* Grant an exemption on access rule DB2LBACWRITEARRAY with the WRITEDOWN option for security policy DATA_ACCESS to user BOBBY.

```
GRANT EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEDOWN
  FOR DATA_ACCESS TO USER BOBBY
```

- *Example 3:* Grant an exemption on access rule DB2LBACWRITEARRAY with the WRITEUP option for security policy DATA_ACCESS to user BOBBY.

```
GRANT EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEUP
  FOR DATA_ACCESS TO USER BOBBY
```

GRANT (global variable privileges)

This form of the GRANT statement grants one or more privileges on a created global variable.

Invocation

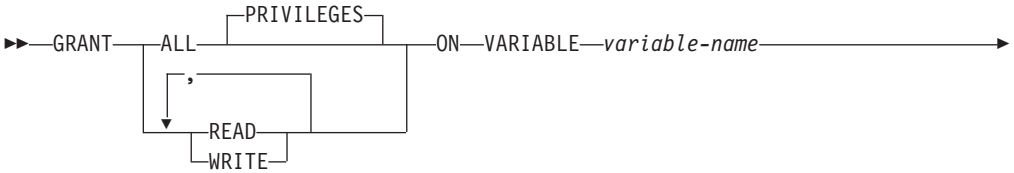
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

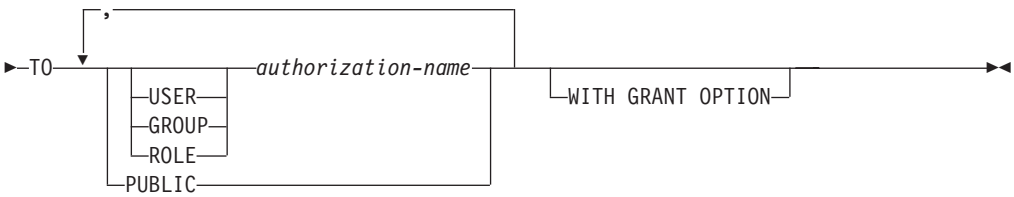
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH GRANT OPTION for each identified privilege on the global variable
- ACCESSCTRL or SECADM authority

Syntax





Description

ALL PRIVILEGES
Grants all privileges on the specified global variable.

READ
Grants the privilege to read the value of the specified global variable.

WRITE
Grants the privilege to assign a value to the specified global variable.

ON VARIABLE *variable-name*
Identifies the global variable on which one or more privileges are to be granted. The *variable-name*, including an implicit or explicit qualifier, must identify a global variable that exists at the current server and is not a module variable (SQLSTATE 42704).

TO Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

ROLE
Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the specified privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges, and object ownership".

WITH GRANT OPTION
Allows the specified *authorization-name* to grant the privileges to others. If the WITH GRANT OPTION clause is omitted, the specified *authorization-name* cannot grant the privileges to others unless that authority has been received from some other source.

Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database and as either GROUP or USER in the operating system, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as both USER and GROUP according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as USER only according to the security plug-in in effect, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined as GROUP only according to the security plug-in in effect, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

Notes

- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Example

Grant the READ and WRITE privilege on global variable MYSCHEMA.MYJOB_PRINTER to user ZUBIRI.

```
GRANT READ, WRITE ON VARIABLE MYSCHEMA.MYJOB_PRINTER TO ZUBIRI
```

GRANT (index privileges)

This form of the GRANT statement grants the CONTROL privilege on indexes.

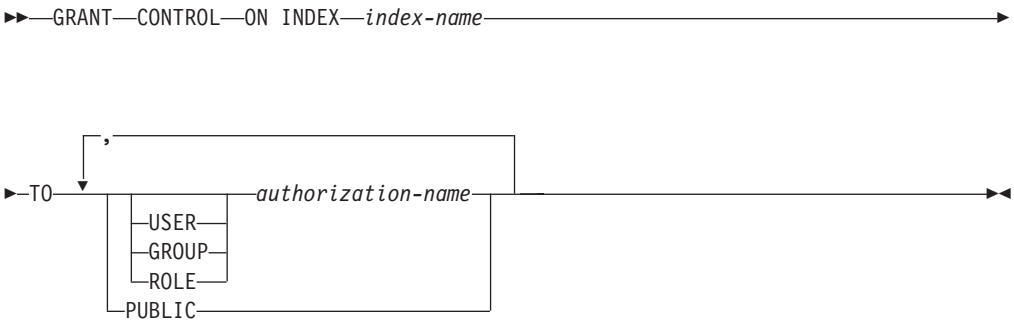
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax



Description

CONTROL
Grants the privilege to drop the index. This is the CONTROL authority for indexes, which is automatically granted to creators of indexes.

ON INDEX *index-name*
Identifies the index for which the CONTROL privilege is to be granted.

TO Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

Notes

- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Example

Grant CONTROL privilege on the DEPTIDX index to the user whose ID is KIESLER:

```
GRANT CONTROL ON INDEX DEPTIDX TO USER KIESLER
```

GRANT (module privileges)

This form of the GRANT statement grants privileges on a module.

Invocation

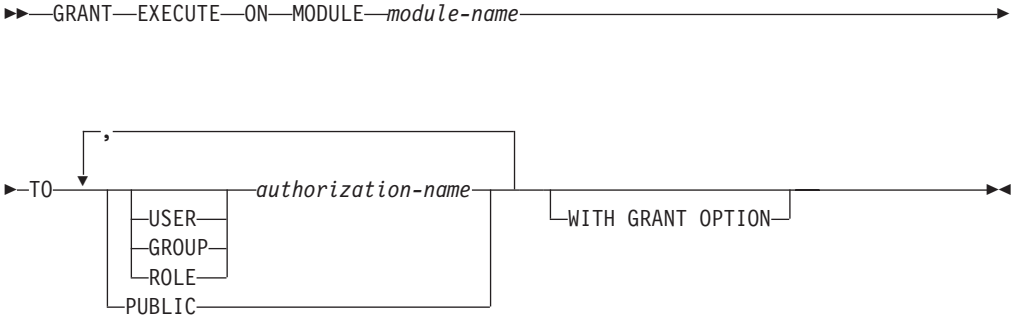
This statement can be embedded in an application program or issued issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH GRANT OPTION for EXECUTE on the module.
- ACCESSCTRL or SECADM authority.

Syntax



Description

EXECUTE
Grants the privilege to reference published module objects. This includes the privilege to:

- Execute any published routines defined in the module.
- Read from and write to any published global variables defined in the module.
- Reference any published user-defined types defined in the module.
- Reference any published conditions defined in the module.

ON MODULE *module-name*
Identifies the module on which the privilege is granted. The module-name must identify a module that exists at the current server (SQLSTATE 42704).

TO Indicates to whom the privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists one or more authorization IDs.

PUBLIC
Grants the privilege to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

WITH GRANT OPTION
Allows the specified *authorization-name*s to grant the EXECUTE privilege to other users. If WITH GRANT OPTION is omitted, the specified *authorization-name*s cannot grant the EXECUTE privilege to others unless they have received that authority from some other source.

Notes

- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Example

Grant the EXECUTE privilege on module MYMODA to user JONES:

```
GRANT EXECUTE
  ON MODULE MYMODA
    TO JONES
```

GRANT (package privileges)

This form of the GRANT statement grants privileges on a package.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

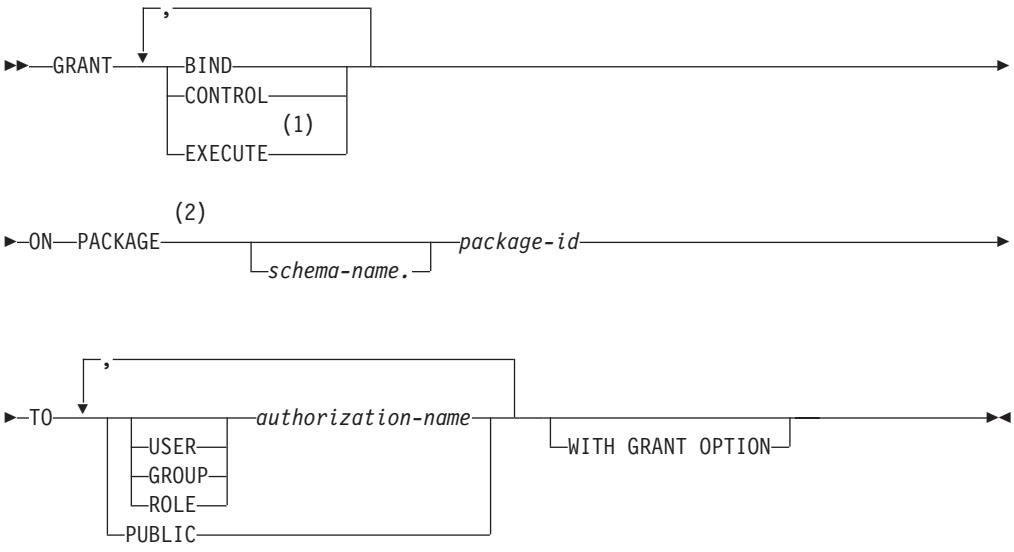
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the referenced package
- The WITH GRANT OPTION for each identified privilege on *package-name*
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to grant the CONTROL privilege.

Syntax



Notes:

1 RUN can be used as a synonym for EXECUTE.

2 PROGRAM can be used as a synonym for PACKAGE.

Description

BIND

Grants the privilege to bind a package. The BIND privilege allows a user to re-issue the BIND command against that package, or to issue the REBIND command. It also allows a user to create a new version of an existing package.

In addition to the BIND privilege, a user must hold the necessary privileges on each table referenced by static DML statements contained in a program. This is necessary, because authorization on static DML statements is checked at bind time.

CONTROL
Grants the privilege to rebind, drop, or execute the package, and extend package privileges to other users. The CONTROL privilege for packages is automatically granted to creators of packages. A package owner is the package binder, or the ID specified with the OWNER option at bind/precompile time.

BIND and EXECUTE are automatically granted to an *authorization-name* that is granted CONTROL privilege.

CONTROL grants the ability to grant the previously mentioned privileges (except for CONTROL) to others.

EXECUTE
Grants the privilege to execute the package.

ON PACKAGE *schema-name.package-id*
Specifies the name of the package on which privileges are to be granted. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The granting of a package privilege applies to all versions of the package (that is, to all packages that share the same package ID and package schema).

TO Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

WITH GRANT OPTION
Allows the specified *authorization-name* to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all of the applicable privileges except for CONTROL (SQLSTATE 01516).

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).

- If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
- If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
- If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
- If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
- If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

Notes

- Package privileges apply to all versions of a package (that is, all packages that share the same package ID and package schema). It is not possible to restrict access to only one version. Because CONTROL privilege is implicitly granted to the binder of a package, if two different users bind two versions of a package, then both users will implicitly be granted access to each other's package.
- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Examples

- *Example 1:* Grant the EXECUTE privilege on PACKAGE CORPDATA.PKGA to PUBLIC.

```
GRANT EXECUTE
  ON PACKAGE CORPDATA.PKGA
  TO PUBLIC
```

- *Example 2:* GRANT EXECUTE privilege on package CORPDATA.PKGA to a user named EMPLOYEE. There is neither a group nor a user called EMPLOYEE.

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO EMPLOYEE
```

or

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO USER EMPLOYEE
```

GRANT (role)

This form of the GRANT statement grants roles to users, groups, or to other roles.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

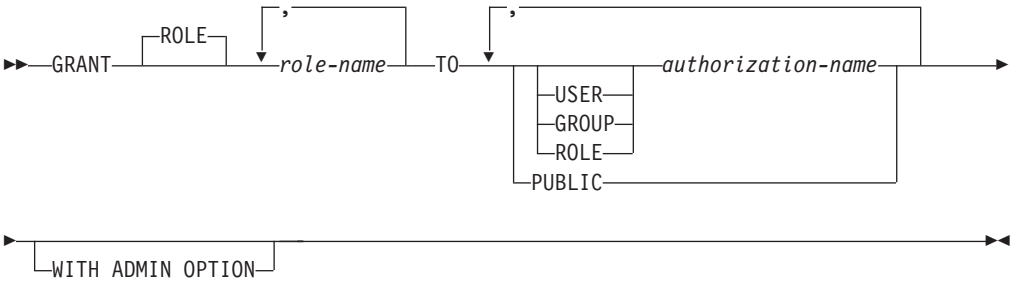
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH ADMIN OPTION on the role
- SECADM authority

SECADM authority is required to grant the WITH ADMIN OPTION to an *authorization-name*.

Syntax



Description

ROLE *role-name*,...
Identifies one or more roles to be granted. Each *role-name* must identify an existing role at the current server (SQLSTATE 42704).

TO Specifies to whom the role is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

ROLE
Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the specified roles to a set of users (authorization IDs). For more information, see "Authorization, privileges, and object ownership".

WITH ADMIN OPTION
Allows the specified *authorization-name* to grant or revoke the *role-name* to or from others, or to associate a comment with the role. It does not allow the specified *authorization-name* to drop the role.

Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database and as either GROUP or USER in the operating system, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as both USER and GROUP according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as USER only according to the security plug-in in effect, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined as GROUP only according to the security plug-in in effect, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- Hierarchies of roles can be built by granting one role to another role. However, cycles are not allowed (SQLSTATE 428GF). For example, if role R1 is granted to another role R2, then role R2 (or some other role R*n* that contains R2) cannot be granted back to R1, because this would produce a cycle.

Notes

- When role R1 is granted to another role R2, then R2 contains R1.
- DBADM authority cannot be granted to PUBLIC. Therefore:
  - Granting role R1 to PUBLIC fails (SQLSTATE 42508) if role R1 holds DBADM authority either directly or indirectly.
    - Role R1 holds DBADM authority directly if the following statement has been issued:

      ```
      GRANT DBADM ON DATABASE TO ROLE R1
      ```

    - Role R1 holds DBADM authority indirectly if the following statements have been issued:

      ```
      GRANT DBADM ON DATABASE TO ROLE R2

      GRANT ROLE R2 TO ROLE R1
      ```

  - Granting role R1, which holds DBADM authority, to role R2 fails (SQLSTATE 42508) if role R2 is granted to PUBLIC either directly or indirectly.
    - Role R2 is granted to PUBLIC directly if the following statement has been issued:

      ```
      GRANT ROLE R2 TO PUBLIC
      ```

    - Role R2 is granted to PUBLIC indirectly if the following statements have been issued:

```
GRANT ROLE R2 TO ROLE R3

GRANT ROLE R3 TO PUBLIC
```

- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Examples

- *Example 1:* Grant role INTERN to role DOCTOR and role DOCTOR to role SPECIALIST.

```
GRANT ROLE INTERN TO ROLE DOCTOR

GRANT ROLE DOCTOR TO ROLE SPECIALIST
```

- *Example 2:* Grant role INTERN to PUBLIC.

```
GRANT ROLE INTERN TO PUBLIC
```

- *Example 3:* Grant role SPECIALIST to user BOB and group TORONTO.

```
GRANT ROLE SPECIALIST TO USER BOB, GROUP TORONTO
```

GRANT (routine privileges)

This form of the GRANT statement grants privileges on a routine (function, method, or procedure) that is not defined in a module.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

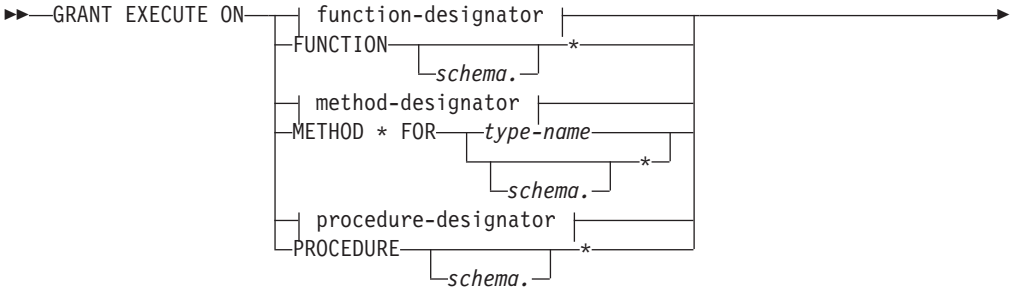
- The WITH GRANT OPTION for EXECUTE on the routine
- ACCESSCTRL or SECADM authority

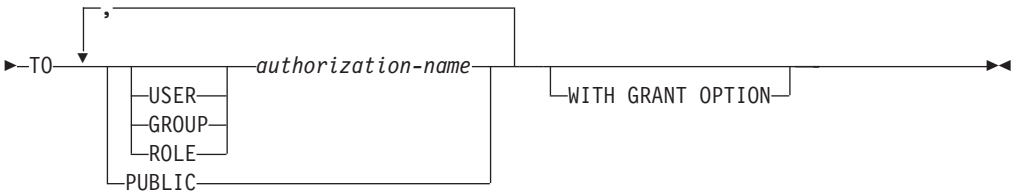
To grant all routine EXECUTE privileges in the schema or type, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH GRANT OPTION for EXECUTE on all existing and future routines (of the specified type) in the specified schema
- ACCESSCTRL or SECADM authority

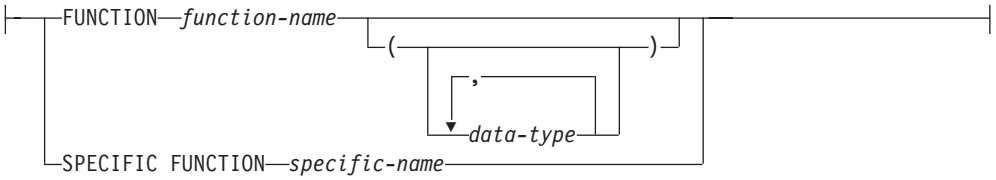
To grant EXECUTE privilege on the audit procedures and table functions SECADM authority is required. EXECUTE privilege WITH GRANT OPTION cannot be granted for these routines (SQLSTATE 42501)

Syntax

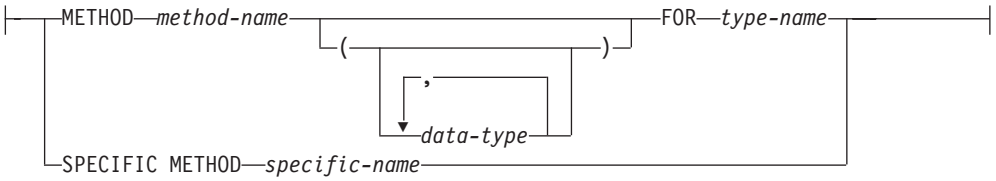




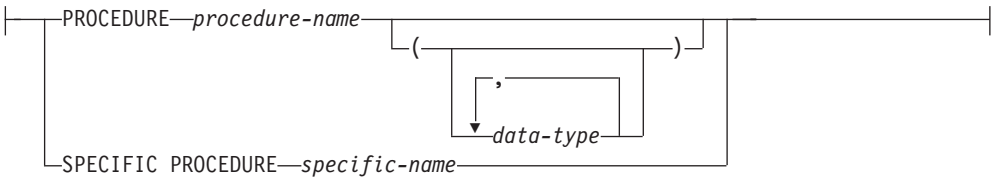
function-designator:



method-designator:



procedure-designator:



Description

EXECUTE
Grants the privilege to run the identified user-defined function, method, or procedure.

function-designator
Uniquely identifies the function on which the privilege is granted. For more information, see "Function, method, and procedure designators" on page 20.

FUNCTION *schema*.*
Identifies all the functions in the schema, including any functions that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

method-designator
Uniquely identifies the method on which the privilege is granted. For more information, see "Function, method, and procedure designators" on page 20.

METHOD *
Identifies all the methods for the type *type-name*, including any methods that may be created in the future.

FOR *type-name*
Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL

statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names. An asterisk (*) can be used in place of *type-name* to identify all types in the schema, including any types that may be created in the future.

procedure-designator
Uniquely identifies the procedure on which the privilege is granted. For more information, see "Function, method, and procedure designators" on page 20.

PROCEDURE *schema*.*
Identifies all the procedures in the schema, including any procedures that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

TO Specifies to whom the EXECUTE privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

PUBLIC
Grants the EXECUTE privilege to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the EXECUTE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the EXECUTE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to grant the EXECUTE privilege from some other source.

Rules

- It is not possible to grant the EXECUTE privilege on a function or method defined with schema 'SYSIBM' or 'SYSFUN' (SQLSTATE 42832).
- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.

  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, and no privileges were granted, a warning is returned (SQLSTATE 01007). If the grantor has no privileges on the object of the grant operation, an error is returned (SQLSTATE 42501).

Notes

- Privileges for a routine defined in a module are granted at the module level using the GRANT (module privileges) statement. The EXECUTE privilege on the module allows access to all objects in the module.
- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Examples

- *Example 1:* Grant the EXECUTE privilege on function CALC_SALARY to user JONES. Assume that there is only one function in the schema with function name CALC_SALARY.

```
GRANT EXECUTE ON FUNCTION CALC_SALARY TO JONES
```

- *Example 2:* Grant the EXECUTE privilege on procedure VACATION_ACCR to all users at the current server.

```
GRANT EXECUTE ON PROCEDURE VACATION_ACCR TO PUBLIC
```

- *Example 3:* Grant the EXECUTE privilege on function DEPT_TOTALS to the administrative assistant and give the assistant the ability to grant the EXECUTE privilege on this function to others. The function has the specific name DEPT85_TOT. Assume that the schema has more than one function named DEPT_TOTALS.

```
GRANT EXECUTE ON SPECIFIC FUNCTION DEPT85_TOT
  TO ADMIN_A WITH GRANT OPTION
```

- *Example 4:* Grant the EXECUTE privilege on function NEW_DEPT_HIRES to HR (Human Resources). The function has two input parameters of type INTEGER and CHAR(10), respectively. Assume that the schema has more than one function named NEW_DEPT_HIRES.

```
GRANT EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10)) TO HR
```

- *Example 5:* Grant the EXECUTE privilege on method SET_SALARY of type EMPLOYEE to user JONES.

```
GRANT EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE TO JONES
```

GRANT (schema privileges)

This form of the GRANT statement grants privileges on a schema.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

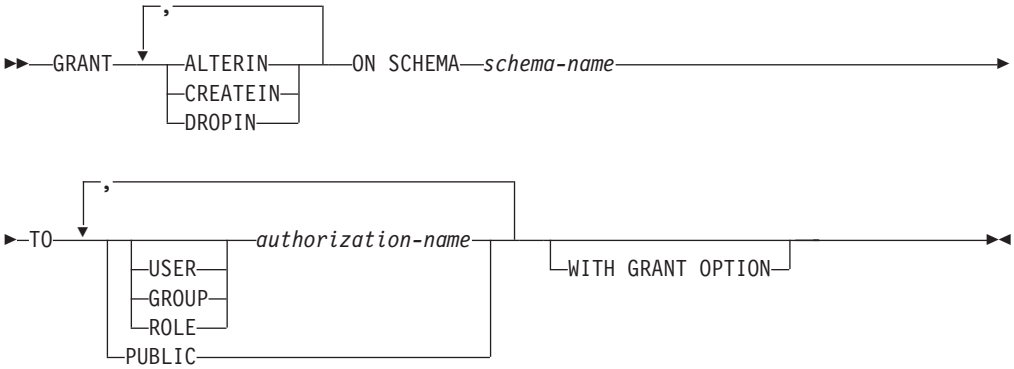
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH GRANT OPTION for each identified privilege on *schema-name*
- ACCESSCTRL or SECADM authority

No user can grant privileges on schema names starting with SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42501).

Syntax



Description

ALTERIN
Grants the privilege to alter or comment on all objects in the schema. The owner of an explicitly created schema automatically receives ALTERIN privilege.

CREATEIN
Grants the privilege to create objects in the schema. Other authorities or privileges required to create the object (such as CREATETAB) are still required. The owner of an explicitly created schema automatically receives CREATEIN privilege. An implicitly created schema has CREATEIN privilege automatically granted to PUBLIC.

DROPIN
Grants the privilege to drop all objects in the schema. The owner of an explicitly created schema automatically receives DROPIN privilege.

ON SCHEMA *schema-name*
Identifies the schema on which the privileges are to be granted.

TO Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E for MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

Notes

- **Grant on SYSPUBLIC:** Privileges can be granted on the reserved schema SYSPUBLIC. Granting CREATEIN privilege allows the user to create a public alias and granting DROPIN privilege allows the user to drop any public alias.
- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package

- A base table while processing a CREATE VIEW statement
- A base table while processing a CREATE TABLE statement for a materialized query table
- Create SQL routine
- Create trigger

Examples

- *Example 1:* Grant user JSINGLETON to the ability to create objects in schema CORPDATA.

```
GRANT CREATEIN ON SCHEMA CORPDATA TO JSINGLETON
```

- *Example 2:* Grant user IHAKES the ability to create and drop objects in schema CORPDATA.

```
GRANT CREATEIN, DROPIN ON SCHEMA CORPDATA TO IHAKES
```

GRANT (security label)

This form of the GRANT statement grants a label-based access control (LBAC) security label to a user, group, or role for read access, write access, or for both read and write access.

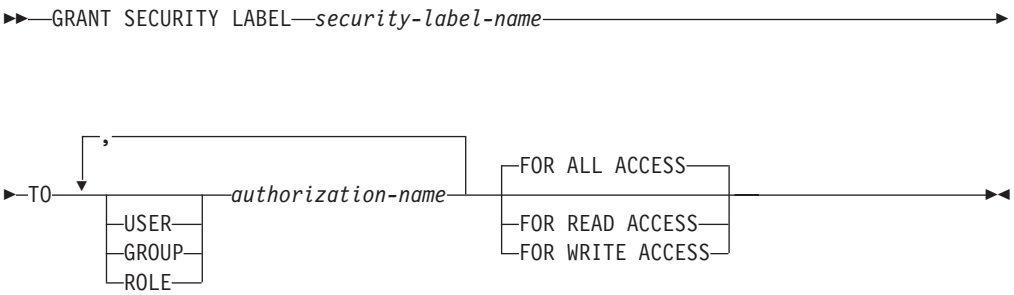
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

SECURITY LABEL *security-label-name*
Grants the security label *security-label-name*. The name must be qualified with a security policy (SQLSTATE 42704) and must identify a security label that exists at the current server (SQLSTATE 42704).

TO Specifies to whom the specified security label is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

FOR ALL ACCESS
Indicates that the security label is to be granted for both read access and write access.

FOR READ ACCESS
Indicates that the security label is to be granted for read access only.

FOR WRITE ACCESS
Indicates that the security label is to be granted for write access only.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- For any given security policy, an *authorization-name* can be granted at most one security label from that policy for read access and one for write access. If the grantee already holds a security label for the type of access (read or write) indicated and that is part of the security policy that qualifies *security-label-name*, an error is returned (SQLSTATE 428GR).
- If the security policy is not defined to consider access through groups or roles, any security label granted to a group or role is ignored when access is attempted.
- If an *authorization-name* holds different security labels for read access and write access, the security labels must meet the following criteria (SQLSTATE 428GQ):
  - If any component in the security labels is of type ARRAY then the value for that component must be the same in both security labels.
  - If any component in the security labels is of type SET then every element in the value for that component in the write security label must also be part of the value for that component in the read security label.
  - If any component in the security labels is of type TREE then every element in the value for that component in the write security label must be the same as or a descendent of one of the elements in the value for that same component in the read security label.

Notes

- By default when a security policy is created, only security labels granted to an individual user are considered. To have groups or roles considered for the security policy, you must issue the ALTER SECURITY POLICY statement and specify USE GROUP AUTHORIZATION or USE ROLE AUTHORIZATION as applicable.

Example

The following statement grants two security labels to user GUYLAINE. The security label EMPLOYEESECLABELREAD is granted for read access and the security label EMPLOYEESECLABELWRITE is granted for write access. Both security labels are part of the security policy DATA_ACCESS.

```
GRANT SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELREAD
  TO USER GUYLAINE FOR READ ACCESS

GRANT SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELWRITE
  TO USER GUYLAINE FOR WRITE ACCESS
```

The same user is now granted the security label BEGINNER for both read and write access. This does not cause an error, because BEGINNER is part of the security policy CLASSPOLICY, and the security labels already held are part of the security policy DATA_ACCESS.

```
GRANT SECURITY LABEL CLASSPOLICY.BEGINNER
  TO USER GUYLAINE FOR ALL ACCESS
```

GRANT (sequence privileges)

This form of the GRANT statement grants privileges on a sequence.

Invocation

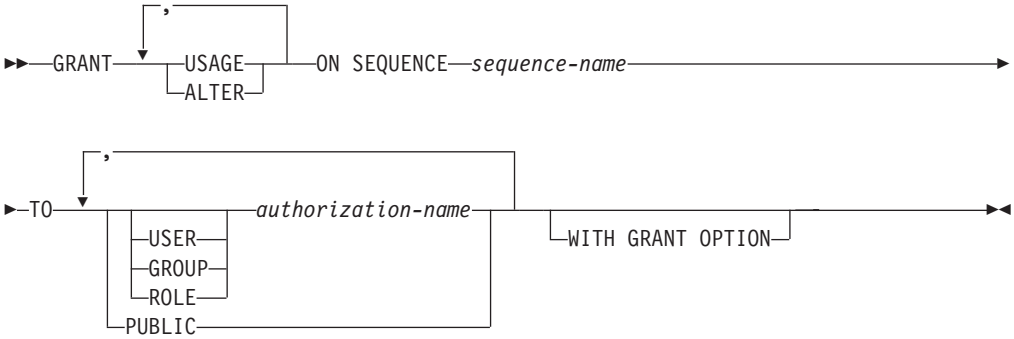
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH GRANT OPTION for each identified privilege on *sequence-name*
- ACCESSCTRL or SECADM authority

Syntax



Description

USAGE
Grants the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

ALTER
Grants the privilege to alter sequence properties using the ALTER SEQUENCE statement.

ON SEQUENCE *sequence-name*
Identifies the sequence on which the specified privileges are to be granted. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error (SQLSTATE 42704) is returned.

TO Specifies to whom the specified privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

PUBLIC
Grants the specified privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

WITH GRANT OPTION
Allows the specified *authorization-name* to grant the specified privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the specified privileges to others if they:

- have SYSADM or DBADM authority or
- received the ability to grant the specified privileges from some other source.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges is not granted. If no privileges are granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

Notes

- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Examples

- *Example 1:* Grant any user the USAGE privilege on a sequence called ORG_SEQ.

```
GRANT USAGE ON SEQUENCE ORG_SEQ TO PUBLIC
```

- *Example 2:* Grant user BOBBY the ability to alter a sequence called GENERATE_ID, and to grant this privilege to others.

```
GRANT ALTER ON SEQUENCE GENERATE_ID TO BOBBY WITH GRANT OPTION
```

GRANT (server privileges)

This form of the GRANT statement grants the privilege to access and use a specified data source in pass-through mode.

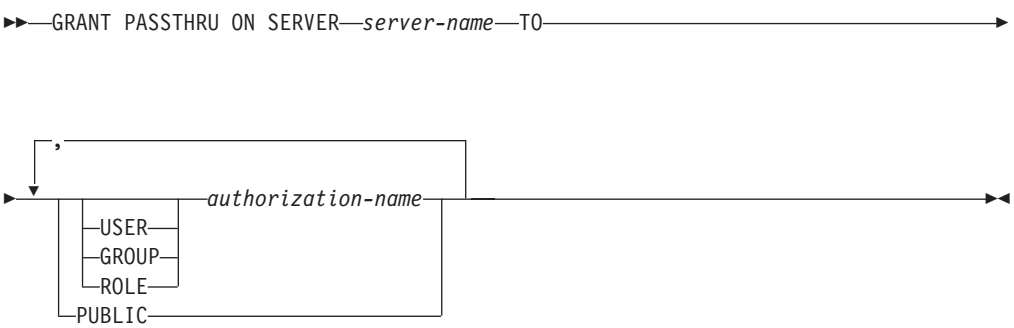
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax



Description

server-name
Names the data source for which the privilege to use in pass-through mode is being granted. *server-name* must identify a data source that is described in the catalog.

TO Specifies to whom the privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants to a set of users (authorization IDs) the privilege to pass through to *server-name*. For more information, see "Authorization, privileges and object ownership".

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

Examples

- *Example 1:* Give R. Smith and J. Jones the privilege to pass through to data source SERVALL. Their authorization IDs are RSMITH and JJONES.

```
GRANT PASSTHRU ON SERVER SERVALL
  TO USER RSMITH,
  USER JJONES
```

- *Example 2:* Grant the privilege to pass through to data source EASTWING to a group whose authorization ID is D024. There is a user whose authorization ID is also D024.

```
GRANT PASSTHRU ON SERVER EASTWING TO GROUP D024
```

  The GROUP keyword must be specified; otherwise, an error will occur because D024 is a user's ID as well as the specified group's ID (SQLSTATE 56092). Any member of group D024 will be allowed to pass through to EASTWING. Therefore, if user D024 belongs to the group, this user will be able to pass through to EASTWING.

GRANT (SETSESSIONUSER privilege)

This form of the GRANT statement grants the SETSESSIONUSER privilege to one or more authorization IDs. The privilege allows the holder to use the SET SESSION AUTHORIZATION statement to set the session authorization to one of a set of specified authorization IDs.

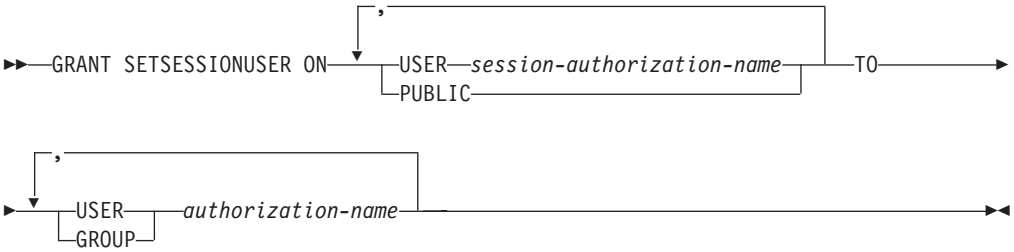
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

SETSESSIONUSER ON
Grants the privilege to assume the identity of a new authorization ID.

USER *session-authorization-name*
Specifies the authorization ID that the *authorization-name* will be able to assume, using the SET SESSION AUTHORIZATION statement. The *session-authorization-name* must identify a user, not a group.

PUBLIC
Specifies that the grantee will be able to assume any valid authorization ID, using the SET SESSION AUTHORIZATION statement.

TO Specifies to whom the privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

Rules

- For each *authorization-name* specified, if neither USER nor GROUP is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.

Notes

- ***Privileges granted to a group***: A privilege that is granted to a group is not used for authorization checking on:
  - Static DML statements in a package
  - A base table while processing a CREATE VIEW statement
  - A base table while processing a CREATE TABLE statement for a materialized query table
  - Create SQL routine
  - Create trigger

Examples

- *Example 1:* The following statement grants user PAUL the ability to set the session authorization to user WALID and therefore to execute statements as WALID.

```
GRANT SETSESSIONUSER ON USER WALID
  TO USER PAUL
```

- *Example 2:* The following statement grants user GUYLAINE the ability to set the session authorization to user BOBBY. It also grants her the ability to set the session authorization to users RICK and KEVIN.

```
GRANT SETSESSIONUSER ON USER BOBBY, USER RICK, USER KEVIN
  TO USER GUYLAINE
```

- *Example 3:* The following statement grants user WALID and everyone in the groups ADMINS and ACCTG the ability to set the session authorization to any user.

```
GRANT SETSESSIONUSER ON PUBLIC TO USER WALID, GROUP ADMINS, ACCTG
```

GRANT (table space privileges)

This form of the GRANT statement grants privileges on a table space.

Invocation

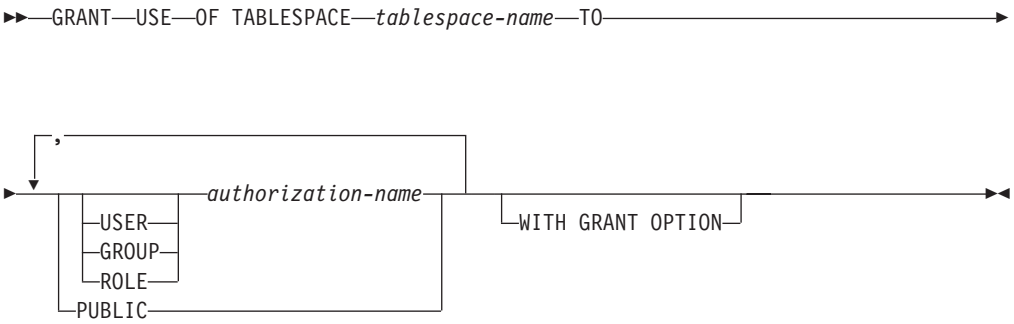
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH GRANT OPTION for use of the table space
- ACCESSCTRL, SECADM, SYSADM, or SYSCTRL authority

Syntax



Description

USE
Grants the privilege to specify or default to the table space when creating a table. The creator of a table space automatically receives USE privilege with grant option.

OF TABLESPACE *tablespace-name*
Identifies the table space on which the USE privilege is to be granted. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a system temporary table space (SQLSTATE 42809).

TO Specifies to whom the USE privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the USE privilege to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

WITH GRANT OPTION
Allows the specified *authorization-name* to GRANT the USE privilege to others.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

Example

Grant user BOBBY the ability to create tables in table space PLANS and to grant this privilege to others.

```
GRANT USE OF TABLESPACE PLANS TO BOBBY WITH GRANT OPTION
```

GRANT (table, view, or nickname privileges)

This form of the GRANT statement grants privileges on a table, view, or nickname.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

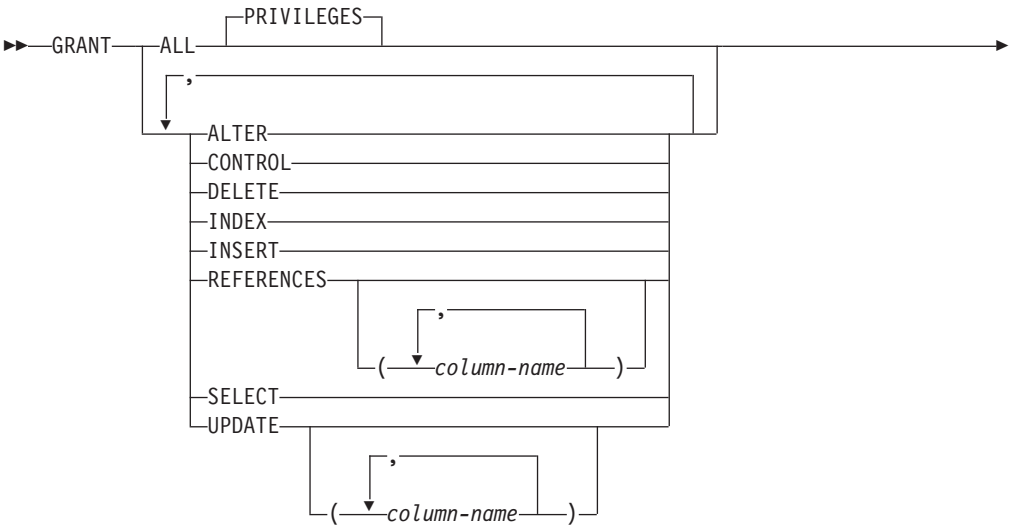
Authorization

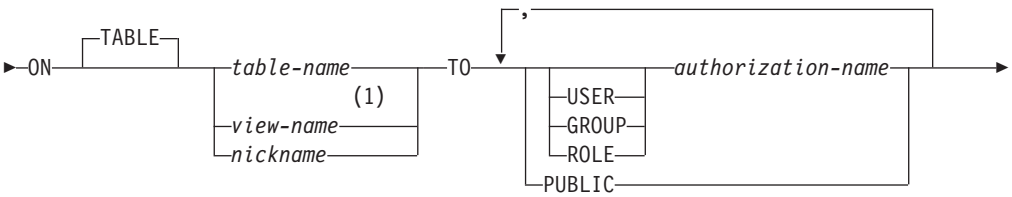
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

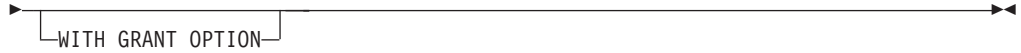
- CONTROL privilege on the referenced table, view, or nickname
- The WITH GRANT OPTION for each identified privilege. If ALL is specified, the authorization ID must have some grantable privilege on the identified table, view, or nickname.
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to grant the CONTROL privilege, or to grant privileges on catalog tables and views.

Syntax







Notes:

1 ALTER, INDEX, and REFERENCES privileges are not applicable to views.

Description

ALL or ALL PRIVILEGES

Grants all the appropriate privileges, except CONTROL, on the base table, view, or nickname named in the ON clause.

If the authorization ID of the statement has CONTROL privilege on the table, view, or nickname, or ACCESSCTRL or SECADM authority, then all the privileges applicable to the object (except CONTROL) are granted. Otherwise, the privileges granted are all those grantable privileges that the authorization ID of the statement has on the identified table, view, or nickname.

If ALL is not specified, one or more of the keywords in the list of privileges must be specified.

ALTER

Grants the privilege to:

- Add columns to a base table definition.
- Create or drop a primary key or unique constraint on a base table.
- Create or drop a foreign key on a base table.

  The REFERENCES privilege on each column of the parent table is also required.
- Create or drop a check constraint on a base table.
- Create a trigger on a base table.
- Add, reset, or drop a column option for a nickname.
- Change a nickname column name or data type.
- Add or change a comment on a base table or a nickname.

CONTROL

Grants:

- All of the appropriate privileges in the list, that is:
  - ALTER, CONTROL, DELETE, INSERT, INDEX, REFERENCES, SELECT, and UPDATE to base tables
  - CONTROL, DELETE, INSERT, SELECT, and UPDATE to views
  - ALTER, CONTROL, INDEX, and REFERENCES to nicknames
- The ability to grant the previously mentioned privileges (except for CONTROL) to others.
- The ability to drop the base table, view, or nickname.

  This ability cannot be extended to others on the basis of holding CONTROL privilege. The only way that it can be extended is by granting the CONTROL privilege itself and that can only be done by an authorization ID with ACCESSCTRL or SECADM authority.
- The ability to execute the RUNSTATS utility on the table and indexes.
- The ability to execute the REORG utility on the table.

- The ability to issue the SET INTEGRITY statement against a base table, materialized query table, or staging table.

The definer of a base table, materialized query table, staging table, or nickname automatically receives the CONTROL privilege.

The definer of a view automatically receives the CONTROL privilege if the definer holds the CONTROL privilege on all tables, views, and nicknames identified in the fullselect.

DELETE
Grants the privilege to delete rows from the table or updatable view.

INDEX
Grants the privilege to create an index on a table, or an index specification on a nickname. This privilege cannot be granted on a view. The creator of an index or index specification automatically has the CONTROL privilege on the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains the CONTROL privilege even if the INDEX privilege is revoked.

INSERT
Grants the privilege to insert rows into the table or updatable view and to run the IMPORT utility.

REFERENCES
Grants the privilege to create and drop a foreign key referencing the table as the parent.

If the authorization ID of the statement has one of:

- ACCESSCTRL or SECADM authority
- CONTROL privilege on the table
- REFERENCES WITH GRANT OPTION on the table

then the grantee(s) can create referential constraints using all columns of the table as parent key, even those added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column REFERENCES privileges that the authorization ID of the statement has on the identified table.

The privilege can be granted on a nickname, although foreign keys cannot be defined to reference nicknames.

REFERENCES (*column-name***,...)**
Grants the privilege to create and drop a foreign key using only those columns specified in the column list as a parent key. Each *column-name* must be an unqualified name that identifies a column of the table identified in the ON clause. Column level REFERENCES privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

SELECT
Grants the privilege to:

- Retrieve rows from the table or view.
- Create views on the table.
- Run the EXPORT utility against the table or view.

UPDATE
Grants the privilege to use the UPDATE statement on the table or updatable view identified in the ON clause.

If the authorization ID of the statement has one of:

- ACCESSCTRL or SECADM authority

- CONTROL privilege on the table or view
- UPDATE WITH GRANT OPTION on the table or view

then the grantee(s) can update all updatable columns of the table or view on which the grantor has with grant privilege as well as those columns added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column UPDATE privileges that the authorization ID of the statement has on the identified table or view.

UPDATE (*column-name***,...)**
Grants the privilege to use the UPDATE statement to update only those columns specified in the column list. Each *column-name* must be an unqualified name that identifies a column of the table or view identified in the ON clause. Column level UPDATE privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

ON TABLE *table-name* **or** *view-name* **or** *nickname*
Specifies the table, view, or nickname on which privileges are to be granted.

No privileges may be granted on an inoperative view or an inoperative materialized query table (SQLSTATE 51024). No privileges may be granted on a declared temporary table (SQLSTATE 42995).

TO Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

A privilege that is granted to a group is not used for authorization checking:

- On static DML statements in a package
- On a base table while processing a CREATE VIEW statement
- On a base table while processing a CREATE TABLE statement for a materialized query table

In DB2 Database for Linux, UNIX, and Windows, table privileges granted to groups only apply to statements that are dynamically prepared. For example, if the INSERT privilege on the PROJECT table has been granted to group D204 but not UBIQUITY (a member of D204) UBIQUITY could issue the statement:

```
EXEC SQL EXECUTE IMMEDIATE :INSERT_STRING;
```

where the content of the string is:

```
INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

but could not precompile or bind a program with the statement:

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

PUBLIC
Grants the privileges to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership". (Previous restrictions on the use of privileges granted to PUBLIC for static SQL statements and the CREATE VIEW statement have been removed.)

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all the applicable privileges except for CONTROL (SQLSTATE 01516).

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database, and as either GROUP or USER according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as both USER and GROUP, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined according to the security plug-in in effect as USER only, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined according to the security plug-in in effect as GROUP only, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.) If CONTROL privilege is specified, privileges will only be granted if the authorization ID of the statement has ACCESSCTRL or SECADM authority (SQLSTATE 42501).

Notes

- Privileges may be granted independently at every level of a table hierarchy. A user with a privilege on a supertable may affect the subtables. For example, an update specifying the supertable *T* may show up as a change to a row in the subtable *S* of *T* done by a user with UPDATE privilege on *T* but without UPDATE privilege on *S*. A user can only operate directly on the subtable if the necessary privilege is held on the subtable.
- Granting nickname privileges has no effect on data source object (table or view) privileges. Typically, data source privileges are required for the table or view that a nickname references when attempting to retrieve data.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. The following syntax is tolerated and ignored:
  - PUBLIC AT ALL LOCATIONS

Examples

- *Example 1:* Grant all privileges on the table WESTERN_CR to PUBLIC.

```
GRANT ALL ON WESTERN_CR
  TO PUBLIC
```

- *Example 2:* Grant the appropriate privileges on the CALENDAR table so that users PHIL and CLAIRE can read it and insert new entries into it. Do not allow them to change or remove any existing entries.

```
GRANT SELECT, INSERT ON CALENDAR
  TO USER  PHIL, USER CLAIRE
```

- *Example 3:* Grant all privileges on the COUNCIL table to user FRANK and the ability to extend all privileges to others.

```
GRANT ALL ON COUNCIL
  TO USER FRANK WITH GRANT OPTION
```

- *Example 4:* GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a user named JOHN. There is a user called JOHN and no group called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

  or

```
GRANT SELECT
  ON CORPDATA.EMPLOYEE TO USER JOHN
```

- *Example 5:* GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a group named JOHN. There is a group called JOHN and no user called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

  or

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO GROUP JOHN
```

- *Example 6:* GRANT INSERT and SELECT on table T1 to both a group named D024 and a user named D024.

```
GRANT INSERT, SELECT ON TABLE T1
  TO GROUP D024, USER D024
```

  In this case, both the members of the D024 group and the user D024 would be allowed to INSERT into and SELECT from the table T1. Also, there would be two rows added to the SYSCAT.TABAUTH catalog view.

- *Example 7:* GRANT INSERT, SELECT, and CONTROL on the CALENDAR table to user FRANK. FRANK must be able to pass the privileges on to others.

```
GRANT CONTROL ON TABLE CALENDAR
  TO FRANK WITH GRANT OPTION
```

  The result of this statement is a warning (SQLSTATE 01516) that CONTROL was not given the WITH GRANT OPTION. Frank now has the ability to grant any privilege on CALENDAR including INSERT and SELECT as required. FRANK cannot grant CONTROL on CALENDAR to other users unless he has ACCESSCTRL or SECADM authority.

- *Example 8:* User JON created a nickname for an Oracle table that had no index. The nickname is ORAREM1. Later, the Oracle DBA defined an index for this table. User SHAWN now wants DB2 to know that this index exists, so that the optimizer can devise strategies to access the table more efficiently. SHAWN can inform DB2 of the index by creating an index specification for ORAREM1. Give SHAWN the index privilege on this nickname, so that he can create the index specification.

```
GRANT INDEX ON NICKNAME ORAREM1
  TO USER SHAWN
```

GRANT (workload privileges)

This form of the GRANT statement grants the USAGE privilege on a workload.

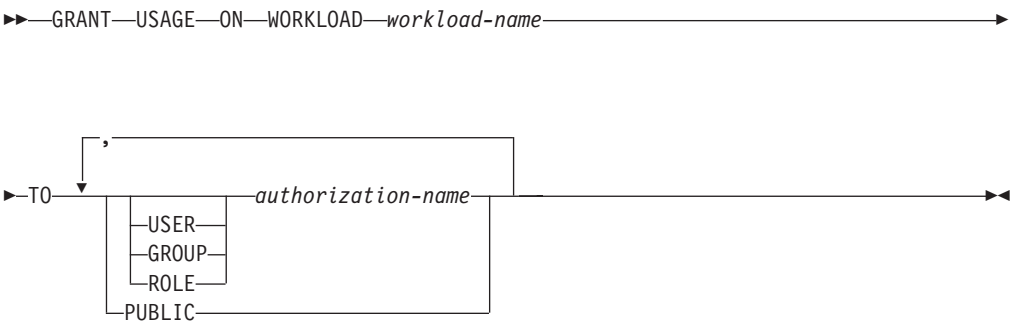
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL, SECADM, or WLMADM authority.

Syntax



Description

USAGE
Grants the privilege to use a workload. Units of work that are submitted by a user will only be mapped to a workload on which the user has USAGE privilege. A user with SYSADM or DBADM authority automatically has USAGE privilege on any workload that exists at the current server.

ON WORKLOAD *workload-name*
Identifies the workload on which the USAGE privilege is to be granted. This is a one-part name. The *workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The name cannot be 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

TO Specifies to whom the USAGE privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

ROLE
Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the USAGE privilege to a set of users (authorization IDs). For more information, see "Authorization, privileges, and object ownership".

Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified:
  - If the security plug-in in effect for the instance cannot determine the status of the *authorization-name*, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as ROLE in the database and as either GROUP or USER in the operating system, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as both USER and GROUP according to the security plug-in in effect, an error is returned (SQLSTATE 56092).
  - If the *authorization-name* is defined as USER only according to the security plug-in in effect, or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined as GROUP only according to the security plug-in in effect, GROUP is assumed.
  - If the *authorization-name* is defined in the database as ROLE only, ROLE is assumed.

Notes

- The GRANT statement does not take effect until it is committed, even for the connection that issues the statement.
- If the database is created with the RESTRICT option, the USAGE privilege of the default user workload, SYSDEFAULTUSERWORKLOAD, must be granted explicitly by a user that has DBADM authority. If the database is created without the RESTRICT option, the USAGE privilege of SYSDEFAULTUSERWORKLOAD is granted to PUBLIC at database creation time.

Example

Grant user LISA the ability to use the workload CAMPAIGN.

```
GRANT USAGE ON WORKLOAD CAMPAIGN TO USER LISA
```

GRANT (XSR object privileges)

This form of the GRANT statement grants USAGE privilege on an XSR object.

Invocation

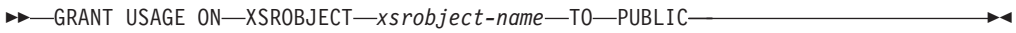
The GRANT statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if the DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

One of the following authorities is required:

- ACCESSCTRL or SECADM authority
- Owner of the XSR object, as recorded in the OWNER column of the SYSCAT.XSROBJECTS catalog view

Syntax



Description

ON XSROBJECT *xsrobject-name*
This name identifies the XSR object for which the USAGE privilege is granted. The *xsrobject-name*, including the implicit or explicit schema qualifier, must uniquely identify an existing XSR object at the current server. If no XSR object by this name exists, an error is returned (SQLSTATE 42704).

TO PUBLIC
Grants the USAGE privilege to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

Example

Grant every user the usage privilege on the XML schema MYSCHEMA:

```
GRANT USAGE ON XSROBJECT MYSCHEMA TO PUBLIC
```

IF

The IF statement selects an execution path based on the evaluation of a condition.

Invocation

This statement can be embedded in an:

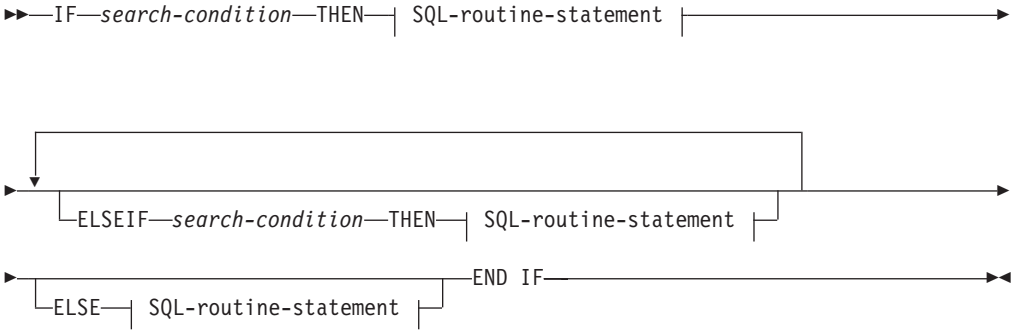
- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

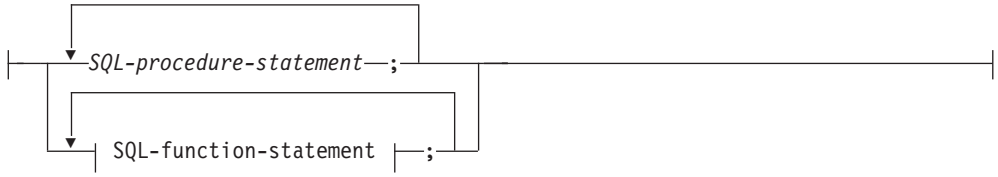
Authorization

Group privileges are not considered because this statement cannot be dynamically prepared.

Syntax



SQL-routine-statement:



Description

search-condition
Specifies the condition for which an SQL statement should be invoked. If the condition is unknown or false, processing continues to the next search condition, until either a condition is true or processing reaches the ELSE clause.

SQL-procedure-statement
Specifies the statement to be invoked if the preceding *search-condition* is true. *SQL-procedure-statement* is only applicable when in the context of an SQL

procedure or a compound SQL (compiled) statement. See *SQL-procedure-statement* in "Compound SQL (compiled)" statement.

SQL-function-statement
Specifies the statement to be invoked if the preceding *search-condition* is true. *SQL-function-statement* is only applicable when in the context of a compound SQL (inlined) statement, an SQL trigger, an SQL function, or an SQL method. See *SQL-function-statement* in "FOR".

Example

The following SQL procedure accepts two IN parameters: an employee number *employee_number* and an employee rating *rating*. Depending on the value of *rating*, the `employee` table is updated with new values in the `salary` and `bonus` columns.

```
CREATE PROCEDURE UPDATE_SALARY_IF
  (IN employee_number CHAR(6), INOUT rating SMALLINT)
  LANGUAGE SQL
  BEGIN
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE EXIT HANDLER FOR not_found
      SET rating = -1;
    IF rating = 1
      THEN UPDATE employee
      SET salary = salary * 1.10, bonus = 1000
      WHERE empno = employee_number;
    ELSEIF rating = 2
      THEN UPDATE employee
      SET salary = salary * 1.05, bonus = 500
      WHERE empno = employee_number;
    ELSE UPDATE employee
      SET salary = salary * 1.03, bonus = 0
      WHERE empno = employee_number;
    END IF;
  END
```

INCLUDE

The INCLUDE statement inserts declarations into a source program.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.

Syntax

Description

SQLCA
Indicates the description of an SQL communication area (SQLCA) is to be included.

SQLDA
Indicates the description of an SQL descriptor area (SQLDA) is to be included.

name
Identifies an external file containing text that is to be included in the source program being precompiled. It can be an SQL identifier without a file name extension or a literal enclosed by single quotation marks (' '). An SQL identifier assumes the filename extension of the source file being precompiled. If a file name extension is not provided by a literal enclosed by quotation marks, none is assumed.

Notes

- When a program is precompiled, the INCLUDE statement is replaced by source statements. Thus, the INCLUDE statement should be specified at a point in the program such that the resulting source statements are acceptable to the compiler.
- The external source file must be written in the host language specified by *name*. If it is greater than 18 bytes or contains characters that are not allowed in an SQL identifier, it must be enclosed by single quotation marks. INCLUDE *name* statements may be nested though not cyclical (for example, if A and B are modules and A contains an INCLUDE *name* statement, then it is not valid for A to call B and then B to call A).
- When the LANGLEVEL precompile option is specified with the SQL92E value, INCLUDE SQLCA should not be specified. SQLSTATE and SQLCODE variables may be defined within the host variable declare section.

Example

Include an SQLCA in a C program.

```
EXEC SQL INCLUDE SQLCA;

EXEC SQL DECLARE C1 CURSOR FOR
```

```
  SELECT DEPTNO, DEPTNAME, MGRNO FROM TDEPT
    WHERE ADMRDEPT = 'A00';

EXEC SQL OPEN C1;

while (SQLCODE==0) {
  EXEC SQL FETCH C1 INTO :dnum, :dname, :mnum;

(Print results)

}

EXEC SQL CLOSE C1;
```

INSERT

The INSERT statement inserts rows into a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect.

Inserting a row into a nickname inserts the row into the data source object to which the nickname refers. Inserting a row into a view also inserts the row into the table on which the view is based, if no INSTEAD OF trigger is defined for the insert operation on this view. If such a trigger is defined, the trigger will be executed instead.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- INSERT privilege on the target table, view, or nickname
- CONTROL privilege on the target table, view, or nickname
- DATAACCESS authority

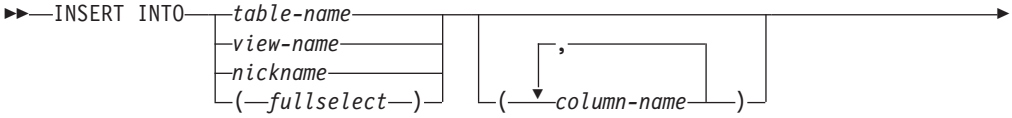
In addition, for each table, view, or nickname referenced in any fullselect used in the INSERT statement, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

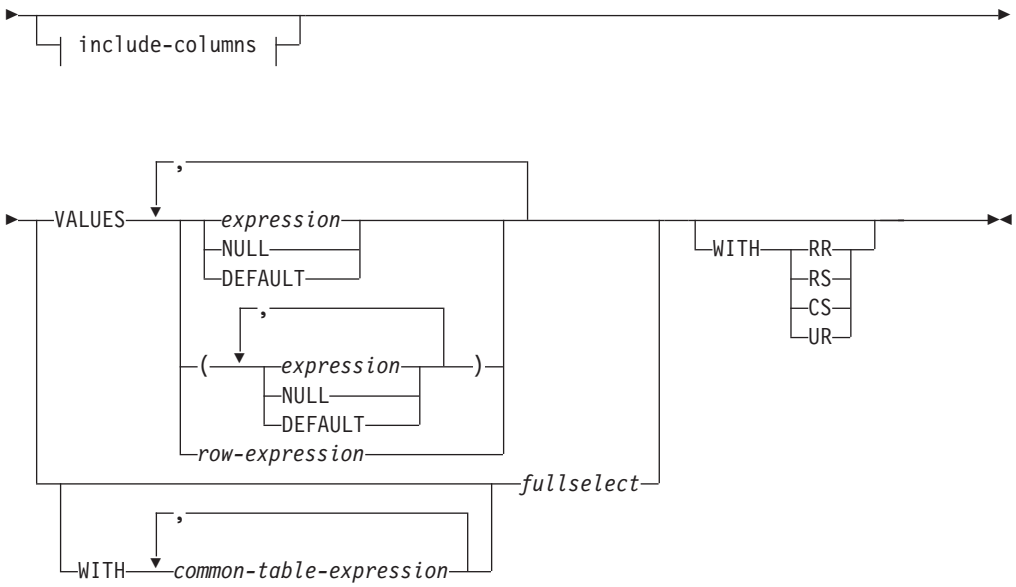
- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

GROUP privileges are not checked for static INSERT statements.

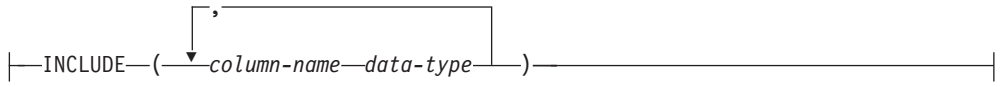
If the target of the insert operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax





include-columns:



Description

INTO *table-name*, *view-name*, *nickname*, **or** *(fullselect)*

Identifies the object of the insert operation. The name must identify one of the following objects:

- A table, view or nickname that exists at the application server
- A table or view at a remote server specified using a remote-object-name

The object must not be a catalog table, a system-maintained materialized query table, a view of a catalog table, or a read-only view, unless an INSTEAD OF trigger is defined for the insert operation on the subject view. Rows inserted into a nickname are placed in the data source object to which the nickname refers.

If the object of the insert operation is a fullselect, the fullselect must be insertable, as defined in the "Insertable views" Notes item in the description of the CREATE VIEW statement.

If the object of the insert operation is a nickname, the extended indicator variable values of DEFAULT and UNASSIGNED must not be used (SQLSTATE 22539).

If no INSTEAD OF trigger exists for the insert operation on this view, a value cannot be inserted into a view column that is derived from the following elements:

- A constant, expression, or scalar function
- The same base table column as some other column of the view

If the object of the insert operation is a view with such columns, a list of column names must be specified, and the list must not identify these columns.

A row can be inserted into a view or a fullselect that is defined using a UNION ALL if the row satisfies the check constraints of exactly one of the underlying base tables. If a row satisfies the check constraints of more than one table, or no table at all, an error is returned (SQLSTATE 23513).

A row cannot be inserted into a view or a fullselect that is defined using a UNION ALL if any base table of the view contains a before trigger and the before trigger contains an UPDATE, a DELETE, or an INSERT operation, or invokes any routine containing such operations (SQLSTATE 42987).

(*column-name***,...)**
Specifies the columns for which insert values are provided. Each name must identify a column of the specified table, view, or nickname, or a column in the fullselect. The same column must not be identified more than once. If extended indicator variables are not enabled, a column that cannot accept inserted values (for example, a column based on an expression) must not be identified.

Omission of the column list is an implicit specification of a list in which every column of the table (that is not implicitly hidden) or view, or every item in the select-list of the fullselect is identified in left-to-right order. This list is established when the statement is prepared and, therefore, does not include columns that were added to a table after the statement was prepared.

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the INSERT statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the INSERT statement. This clause can only be specified if the INSERT statement is nested in the FROM clause of a fullselect.

column-name
Specifies a column of the intermediate result table of the INSERT statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

VALUES
Introduces one or more rows of values to be inserted.

Each row specified in the VALUES clause must be assignable to the implicit or explicit column list and the columns identified in the INCLUDE clause, unless a row variable is used. When a row value list in parentheses is specified, the first value is inserted into the first column in the list, the second value into the second column, and so on. When a row expression is specified, the number of fields in the row type must match the number of names in the implicit or explicit column list.

expression
An *expression* can be any expression defined in the “Expressions” topic. If *expression* is a row type, it must not appear in parentheses. If *expression* is a variable, the host variable can include an indicator variable or in the case of a host structure, an indicator array, enabled for extended indicator variables. If extended indicator variables are enabled, the extended

indicator variable values of default (-5) or unassigned (-7) must not be used (SQLSTATE 22539) if either of the following statements is true:

- The expression is more complex than a single host variable with explicit casts
- The target column has data type of structured type

NULL
Specifies the null value and should only be specified for nullable columns.

DEFAULT
Specifies that the default value is to be used. The result of specifying DEFAULT depends on how the column was defined, as follows:

- If the column was defined as a generated column based on an expression, the column value is generated by the system, based on that expression.
- If the IDENTITY clause is used, the value is generated by the database manager.
- If the ROW CHANGE TIMESTAMP clause is used, the value for each inserted row is generated by the database manager as a timestamp that is unique for the table partition within the database partition.
- If the WITH DEFAULT clause is used, the value inserted is as defined for the column (see *default-clause* in "CREATE TABLE").
- If the NOT NULL clause is used and the GENERATED clause is not used, or the WITH DEFAULT clause is not used or DEFAULT NULL is used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).
- When inserting into a nickname, the DEFAULT keyword will be passed through the INSERT statement to the data source only if the data source supports the DEFAULT keyword in its query language syntax.

row-expression
Specifies any row expression of the type described in "Row expressions" that does not include a column name. The number of fields in the row must match the target of the insert and each field must be assignable to the corresponding column.

WITH *common-table-expression*
Defines a common table expression for use with the fullselect that follows.

fullselect
Specifies a set of new rows in the form of the result table of a fullselect. There may be one, more than one, or none. If the result table is empty, SQLCODE is set to +100 and SQLSTATE is set to '02000'.

When the base object of the INSERT and the base object of the fullselect or any subquery of the fullselect, are the same table, the fullselect is completely evaluated before any rows are inserted.

The number of columns in the result table must equal the number of names in the column list. The value of the first column of the result is inserted in the first column in the list, the second value in the second column, and so on.

If the expression that specifies the value of a result column is a variable, the host variable can include an indicator variable enabled for extended indicator variables. If extended indicator variables are enabled, and the expression is more than a single host variable, or a host variable being explicitly cast, then the extended indicator variable values of default or unassigned must not be

used (SQLSTATE 22539). The effects of default or unassigned values apply to the corresponding target columns of the *fullselect*.

WITH
Specifies the isolation level at which the fullselect is executed.

RR Repeatable Read

RS Read Stability

CS Cursor Stability

UR Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

Rules

- ***Triggers:*** INSERT statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the inserted values. If an insert operation into a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.
- ***Default values:*** The value inserted in any column that is not in the column list is either the default value of the column or null. Columns that do not allow null values and are not defined with NOT NULL WITH DEFAULT must be included in the column list. Similarly, if you insert into a view, the value inserted into any column of the base table that is not in the view is either the default value of the column or null. Hence, all columns of the base table that are not in the view must have either a default value or allow null values. The only value that can be inserted into a generated column defined with the GENERATED ALWAYS clause is DEFAULT (SQLSTATE 428C9).
- ***Length:*** If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must either be a string column with a length attribute at least as great as the length of the string, or a datetime column if the string represents a date, time, or timestamp.
- ***Assignment:*** Insert values are assigned to columns in accordance with specific assignment rules.
- ***Validity:*** If the table named, or the base table of the view named, has one or more unique indexes, each row inserted into the table must conform to the constraints imposed by those indexes. If a view whose definition includes WITH CHECK OPTION is named, each row inserted into the view must conform to the definition of the view. For an explanation of the rules governing this situation, see “CREATE VIEW”.
- ***Referential integrity:*** For each constraint defined on a table, each non-null insert value of the foreign key must be equal to a primary key value of the parent table.
- ***Check constraint:*** Insert values must satisfy the check conditions of the check constraints defined on the table. An INSERT to a table with check constraints defined has the constraint conditions evaluated once for each row that is inserted.
- ***XML values:*** A value that is inserted into an XML column must be a well-formed XML document (SQLSTATE 2200M).

- *Security policy:* If the identified table or the base table of the identified view is protected with a security policy, the session authorization ID must have the label-based access control (LBAC) credentials that allow:
  - Write access to all protected columns for which a data value is explicitly provided (SQLSTATE 42512)
  - Write access for any explicit value provided for a DB2SECURITYLABEL column for security policies that were created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option (SQLSTATE 23523)

  The session authorization ID must also have been granted a security label for write access for the security policy if an implicit value is used for a DB2SECURITYLABEL column (SQLSTATE 23523), which can happen when:
  - A value for the DB2SECURITYLABEL column is not explicitly provided
  - A value for the DB2SECURITYLABEL column is explicitly provided but the session authorization ID does not have write access for that value, and the security policy is created with the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option
- **Extended indicator variable usage**: If enabled, negative indicator variable values outside the range of -1 through -7 must not be input (SQLSTATE 22010). Also, if enabled, the default and unassigned extended indicator variable values must not appear in contexts in which they are not supported (SQLSTATE 22539).
- **Extended indicator variables**: In an INSERT statement, a value of unassigned has the effect of setting the column to its default value.

  If the target column is a column defined as GENERATED ALWAYS, then it must be assigned the DEFAULT keyword, or the extended indicator variable-based values of default or unassigned (SQLSTATE 428C9).

Notes

- After execution of an INSERT statement, the value of the third variable of the SQLERRD(3) portion of the SQLCA indicates the number of rows that were passed to the insert operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) contains the count of all triggered insert, update and delete operations.
- Unless appropriate locks already exist, one or more exclusive locks are acquired at the execution of a successful INSERT statement. Until the locks are released, an inserted row can only be accessed by:
  - The application process that performed the insert.
  - Another application process using isolation level UR through a read-only cursor, SELECT INTO statement, or subselect used in a subquery.
- For further information about locking, see the description of the COMMIT, ROLLBACK, and LOCK TABLE statements.
- If an application is running against a partitioned database, and it is bound with option INSERT BUF, then INSERT with VALUES statements which are not processed using EXECUTE IMMEDIATE may be buffered. DB2 assumes that such an INSERT statement is being processed inside a loop in the application's logic. Rather than execute the statement to completion, it attempts to buffer the new row values in one or more buffers. As a result the actual insertions of the rows into the table are performed later, asynchronous with the application's INSERT logic. Be aware that this asynchronous insertion may cause an error related to an INSERT to be returned on some other SQL statement that follows the INSERT in the application.

This has the potential to dramatically improve INSERT performance, but is best used with clean data, due to the asynchronous nature of the error handling.

- When a row is inserted into a table that has an identity column, DB2 generates a value for the identity column.
  - For a GENERATED ALWAYS identity column, DB2 always generates the value.
  - For a GENERATED BY DEFAULT column, if a value is not explicitly specified (with a VALUES clause, or subselect), DB2 generates a value.

  The first value generated by DB2 is the value of the START WITH specification for the identity column.
- When a value is inserted for a user-defined distinct type identity column, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type before the computation.)
- When inserting into a GENERATED ALWAYS identity column, DB2 will always generate a value for the column, and users must not specify a value at insertion time. If a GENERATED ALWAYS identity column is listed in the column-list of the INSERT statement, with a non-DEFAULT value in the VALUES clause, an error occurs (SQLSTATE 428C9).

  For example, assuming that EMPID is defined as an identity column that is GENERATED ALWAYS, then the command:

```
INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
  VALUES (:hv_valid_emp_id, :hv_name, :hv_addr)
```

  will result in an error.
- When inserting into a GENERATED ALWAYS ROW CHANGE TIMESTAMP column, DB2 will always generate a value for the column, and users must not specify a value at insertion time (SQLSTATE 428C9) . The value generated by DB2 is unique for each row inserted on the database partition.
- When inserting into a GENERATED BY DEFAULT column, DB2 will allow an actual value for the column to be specified within the VALUES clause, or from a subselect. However, when a value is specified in the VALUES clause, DB2 does not perform any verification of the value. To guarantee uniqueness of IDENTITY column values, a unique index on the identity column must be created.

  When inserting into a table with a GENERATED BY DEFAULT identity column, without specifying a column list, the VALUES clause can specify the DEFAULT keyword to represent the value for the identity column. DB2 will generate the value for the identity column.

```
INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
  VALUES (DEFAULT, :hv_name, :hv_addr)
```

  In this example, EMPID is defined as an identity column, and thus the value inserted into this column is generated by DB2.
- The rules for inserting into an identity column with a subselect are similar to those for an insert with a VALUES clause. A value for an identity column may only be specified if the identity column is defined as GENERATED BY DEFAULT.

  For example, assume T1 and T2 are tables with the same definition, both containing columns *intcol1* and *identcol2* (both are type INTEGER and the second column has the identity attribute). Consider the following insert:

```
INSERT INTO T2
  SELECT *
  FROM T1
```

This example is logically equivalent to:

```
INSERT INTO T2 (intcol1,identcol2)
  SELECT intcol1, identcol2
  FROM T1
```

In both cases, the INSERT statement is providing an explicit value for the identity column of T2. This explicit specification can be given a value for the identity column, but the identity column in T2 must be defined as GENERATED BY DEFAULT. Otherwise, an error will result (SQLSTATE 428C9).

If there is a table with a column defined as a GENERATED ALWAYS identity, it is still possible to propagate all other columns from a table with the same definition. For example, given the example tables T1 and T2 described previously, the intcol1 values from T1 to T2 can be propagated with the following SQL:

```
INSERT INTO T2 (intcol1)
  SELECT intcol1
  FROM T1
```

Note that, because identcol2 is not specified in the column-list, it will be filled in with its default (generated) value.

- When inserting a row into a single column table where the column is defined as a GENERATED ALWAYS identity column or a ROW CHANGE TIMESTAMP column, it is possible to specify a VALUES clause with the DEFAULT keyword. In this case, the application does not provide any value for the table, and DB2 generates the value for the identity or ROW CHANGE TIMESTAMP column.

  ```
  INSERT INTO IDTABLE
    VALUES(DEFAULT)
  ```

  Assuming the same single column table for which the column has the identity attribute, to insert multiple rows with a single INSERT statement, the following INSERT statement could be used:

  ```
  INSERT INTO IDTABLE
    VALUES (DEFAULT), (DEFAULT), (DEFAULT), (DEFAULT)
  ```

- When DB2 generates a value for an identity column, that generated value is consumed; the next time that a value is needed, DB2 will generate a new value. This is true even when an INSERT statement involving an identity column fails or is rolled back.

  For example, assume that a unique index has been created on the identity column. If a duplicate key violation is detected in generating a value for an identity column, an error occurs (SQLSTATE 23505) and the value generated for the identity column is considered to be consumed. This can occur when the identity column is defined as GENERATED BY DEFAULT and the system tries to generate a new value, but the user has explicitly specified values for the identity column in previous INSERT statements. Reissuing the same INSERT statement in this case can lead to success. DB2 will generate the next value for the identity column, and it is possible that this next value will be unique, and that this INSERT statement will be successful.

- If the maximum value for the identity column is exceeded (or minimum value for a descending sequence) in generating a value for an identity column, an error occurs (SQLSTATE 23522). In this situation, the user would have to DROP and CREATE a new table with an identity column having a larger range (that is, change the data type or increment value for the column to allow for a larger range of values).

  For example, an identity column may have been defined with a data type of SMALLINT, and eventually the column runs out of assignable values. To

redefine the identity column as INTEGER, the data would need to be unloaded, the table would have to be dropped and recreated with a new definition for the column, and then the data would be reloaded. When the table is redefined, it needs to specify a START WITH value for the identity column such that the next value generated by DB2 will be the next value in the original sequence. To determine the end value, issue a query using MAX of the identity column (for an ascending sequence), or MIN of the identity column (for a descending sequence), before unloading the data.

- **Extended indicator variables and insert triggers**: No change in the activation of insert triggers results from use of extended indicator variables. If all columns in the implicit or explicit column list have been assigned to an extended indicator variable-based value of unassigned or default, an insert where all columns have their respective default values is attempted, and if successful, the insert trigger is activated.
- **Extended indicator variables and deferred error checks**: When extended indicator variables are enabled, validation that would otherwise be done in statement preparation, to recognize an insert into a non-updatable column, is deferred until statement execution. Whether an error should be reported can be determined only during execution.
- **Inserting into tables with row-begin, row-end, or transaction start-ID columns**: When a row is inserted into a table with these generated columns (for instance, a system-period temporal table), the database manager assigns values to the following columns:
  - A row-begin column is assigned a value that is generated using a reading of the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to the row-begin or transaction start-ID column in a table, or a row in a system-period temporal table is deleted. The database manager ensures uniqueness of the generated values for a row-begin column across transactions. If multiple rows are inserted within a single SQL transaction, the values for the row-begin column are the same for all the rows and are unique from the values generated for the column for another transaction.
  - A row-end column is assigned the maximum value for the data type of the column (9999-12-30-00.00.00.000000000000).
  - A transaction start-ID column is assigned a unique timestamp value per transaction or the null value. The null value is assigned to the transaction start-ID column if the column is nullable. Otherwise, the value is generated using a reading of the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to the row-begin or transaction start-ID column in a table, or a row in a system-period temporal table is deleted. If multiple rows are inserted within a single SQL transaction, the values for the transaction start-ID column are the same for all the rows and are unique from the values generated for the column for another transaction.
- **Inserting into a system-period temporal table**: When a row is inserted into a system-period temporal table, the database manager assigns values to columns as indicated for tables with row-begin, row-end, or transaction start-ID columns. Also, when a row is inserted, no rows are added to the history table associated with the system-period temporal table.
- **Inserting into application-period temporal tables**: An error is returned when a row is inserted into an application-period temporal table and the following conditions are met:

  - The application-period temporal table has either a primary key or unique constraint with the BUSINESS_TIME WITHOUT OVERLAPS clause defined, or a unique index with the BUSINESS_TIME WITHOUT OVERLAPS clause defined.
  - The period defined by the begin and end columns of the BUSINESS_TIME period overlap the period defined by the begin and end columns of the BUSINESS_TIME period for another row that matches the other columns of the same unique constraint or unique index.
- **Considerations for an INSERT without a column list**: An INSERT statement without a column list does not include implicitly hidden columns. Columns that are defined as implicitly hidden and not null must have a defined default value.

Examples

- *Example 1:* Insert a new department with the following specifications into the DEPARTMENT table:
  - Department number (DEPTNO) is 'E31'
  - Department name (DEPTNAME) is 'ARCHITECTURE'
  - Managed by (MGRNO) a person with number '00390'
  - Reports to (ADMRDEPT) department 'E01'.

```
INSERT INTO DEPARTMENT
  VALUES ('E31', 'ARCHITECTURE', '00390', 'E01')
```

- *Example 2:* Insert a new department into the DEPARTMENT table as in example 1, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT )
  VALUES ('E31', 'ARCHITECTURE', 'E01')
```

- *Example 3:* Insert two new departments using one statement into the DEPARTMENT table as in example 2, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
  VALUES ('B11', 'PURCHASING', 'B01'),
         ('E41', 'DATABASE ADMINISTRATION', 'E01')
```

- *Example 4:* Create a temporary table MA_EMP_ACT with the same columns as the EMP_ACT table. Load MA_EMP_ACT with the rows from the EMP_ACT table with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE TABLE MA_EMP_ACT
         ( EMPNO CHAR(6)  NOT NULL,
          PROJNO CHAR(6)  NOT NULL,
          ACTNO SMALLINT  NOT NULL,
          EMPTIME DEC(5,2),
          EMSTDATE DATE,
          EMENDATE  DATE )
INSERT INTO MA_EMP_ACT
  SELECT * FROM EMP_ACT
    WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

- *Example 5:* Use a C program statement to add a skeleton project to the PROJECT table. Obtain the project number (PROJNO), project name (PROJNAME), department number (DEPTNO), and responsible employee (RESPEMP) from host variables. Use the current date as the project start date (PRSTDATE). Assign a null value to the remaining columns in the table.

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE)
  VALUES (:PRJNO, :PRJNM, :DPTNO, :REMP, CURRENT DATE);
```

- *Example 6:* Specify an INSERT statement as the *data-change-table-reference* within a SELECT statement. Define an extra include column whose values are specified in the VALUES clause, which is then used as an ordering column for the inserted rows.

```
SELECT INORDER.ORDERNUM
  FROM NEW TABLE (INSERT INTO ORDERS(CUSTNO)INCLUDE (INSERTNUM INTEGER)
    VALUES(:CNUM1, 1), (:CNUM2, 2)) InsertedOrders
  ORDER BY INSERTNUM;
```

- *Example 7:* Use a C program statement to add a document to the DOCUMENTS table. Obtain values for the document ID (DOCID) column and the document data (XMLDOC) column from a host variable that binds to an SQL TYPE IS XML AS BLOB_FILE.

```
EXEC SQL INSERT INTO DOCUMENTS
  (DOCID, XMLDOC) VALUES (:docid, :xmldoc)
```

- *Example 8:* For the following INSERT statements, assume that table SALARY_INFO is defined with three columns, and that the last column is an implicitly hidden ROW CHANGE TIMESTAMP column. In the following statement, the implicitly hidden column is explicitly referenced in the column list and a value is provided for it in the VALUES clause.

```
INSERT INTO SALARY_INFO (LEVEL, SALARY, UPDATE_TIME)
  VALUES (2, 30000, CURRENT TIMESTAMP)
```

  The following INSERT statement uses an implicit column list. An implicit column list does not include implicitly hidden columns, so the VALUES clause only contains values for the other two columns.

```
INSERT INTO SALARY_INFO VALUES (2, 30000)
```

  In this case, the UPDATE_TIME column must be defined to have a default value, and that default value is used for the row that is inserted.

ITERATE

The ITERATE statement causes the flow of control to return to the beginning of a labelled loop.

Invocation

This statement can be embedded in an:

- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

Authorization

None required.

Syntax

Description

label
Specifies the label of the FOR, LOOP, REPEAT, or WHILE statement to which DB2 passes the flow of control.

Example

This example uses a cursor to return information for a new department. If the *not_found* condition handler was invoked, the flow of control passes out of the loop. If the value of *v_dept* is 'D11', an ITERATE statement passes the flow of control back to the top of the LOOP statement. Otherwise, a new row is inserted into the DEPARTMENT table.

```
CREATE PROCEDURE ITERATOR()
  LANGUAGE SQL
  BEGIN
    DECLARE v_dept CHAR(3);
    DECLARE v_deptname VARCHAR(29);
    DECLARE v_admdept CHAR(3);
    DECLARE at_end INTEGER DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
      SELECT deptno, deptname, admrdept
      FROM department
      ORDER BY deptno;
    DECLARE CONTINUE HANDLER FOR not_found
      SET at_end = 1;
    OPEN c1;
    ins_loop:
    LOOP
      FETCH c1 INTO v_dept, v_deptname, v_admdept;
      IF at_end = 1 THEN
        LEAVE ins_loop;
      ELSEIF v_dept = 'D11' THEN
```

```
      ITERATE ins_loop;
    END IF;
    INSERT INTO department (deptno, deptname, admrdept)
    VALUES ('NEW', v_deptname, v_admdept);
  END LOOP;
  CLOSE c1;
END
```

LEAVE

The LEAVE statement transfers program control out of a loop or a compound statement.

Invocation

This statement can be embedded in an:

- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

Authorization

None required.

Syntax

Description

label
Specifies the label of the compound, FOR, LOOP, REPEAT, or WHILE statement to exit.

Notes

- When a LEAVE statement transfers control out of a compound statement, all open cursors in the compound statement, except cursors that are used to return result sets, are closed.

Example

This example contains a loop that fetches data for cursor *c1*. If the value of SQL variable *at_end* is not zero, the LEAVE statement transfers control out of the loop.

```
CREATE PROCEDURE LEAVE_LOOP(OUT counter INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE v_counter INTEGER;
    DECLARE v_firstnme VARCHAR(12);
    DECLARE v_midinit CHAR(1);
    DECLARE v_lastname VARCHAR(15);
    DECLARE at_end SMALLINT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
      SELECT firstnme, midinit, lastname
        FROM employee;
    DECLARE CONTINUE HANDLER for not_found
      SET at_end = 1;
    SET v_counter = 0;
    OPEN c1;
    fetch_loop:
    LOOP
      FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
```

```
    IF at_end <> 0 THEN LEAVE fetch_loop;
    END IF;
    SET v_counter = v_counter + 1;
  END LOOP fetch_loop;
  SET counter = v_counter;
  CLOSE c1;
END
```

LOCK TABLE

The LOCK TABLE statement prevents concurrent application processes from using or changing a table. The lock is released when the unit of work issuing the LOCK TABLE statement either commits or terminates.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- SELECT privilege on the table
- CONTROL privilege on the table
- DATAACCESS authority

Syntax

Description

table-name **or** *nickname*
Identifies the table or nickname. The *table-name* must identify a table that exists at the application server, but it must not identify a catalog table, a created temporary table, or a declared temporary table (SQLSTATE 42995). If the *table-name* is a typed table, it must be the root table of the table hierarchy (SQLSTATE 428DR). When a nickname is specified, DB2 will lock the underlying object (that is, a table or view) of the data source to which the nickname refers.

IN SHARE MODE
Prevents concurrent application processes from executing any but read-only operations on the table.

IN EXCLUSIVE MODE
Prevents concurrent application processes from executing any operations on the table. Note that EXCLUSIVE MODE does not prevent concurrent application processes that are running at isolation level Uncommitted Read (UR) from executing read-only operations on the table.

Notes

- Locking is used to prevent concurrent operations. A lock is not necessarily acquired during execution of the LOCK TABLE statement if a suitable lock already exists. The lock that prevents concurrent operations is held at least until termination of the unit of work.
- In a partitioned database, a table lock is first acquired at the first database partition in the database partition group (the database partition with the lowest number) and then at other database partitions. If the LOCK TABLE statement is interrupted, the table may be locked on some database partitions but not on

others. If this occurs, either issue another LOCK TABLE statement to complete the locking on all database partitions, or issue a COMMIT or ROLLBACK statement to release the current locks.

- This statement affects all database partitions in the database partition group.
- For partitioned tables, the only lock acquired for the LOCK TABLE statement is at the table level; no data partition locks are acquired.

Example

Obtain a lock on the table EMP. Do not allow other programs to read or update the table.

```
LOCK TABLE EMP IN EXCLUSIVE MODE
```

LOOP

The LOOP statement repeats the execution of a statement or a group of statements.

Invocation

This statement can be embedded in an:

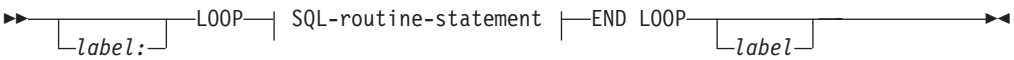
- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

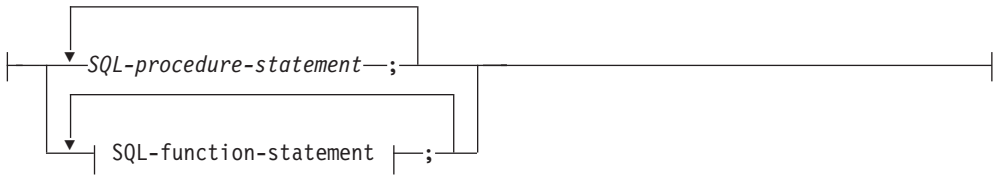
Authorization

No privileges are required to invoke the LOOP statement. However, the authorization ID of the statement must hold the necessary privileges to invoke the SQL statements that are embedded in the LOOP statement.

Syntax



SQL-routine-statement:



Description

label
Specifies the label for the LOOP statement. If the beginning label is specified, that label can be specified on LEAVE and ITERATE statements. If the ending label is specified, a matching beginning label must be specified.

SQL-procedure-statement
Specifies the SQL statements that are to be invoked in the loop. *SQL-procedure-statement* is only applicable when in the context of an SQL procedure or Compound SQL (compiled) statement. See *SQL-procedure-statement* in "Compound SQL (compiled)" statement.

SQL-function-statement
Specifies the SQL statements that are to be invoked in the loop. *SQL-function-statement* is only applicable when in the context of an SQL function, SQL method, or Compound SQL (inlined) statement. See *SQL-function-statement* in "FOR".

Example

This procedure uses a LOOP statement to fetch values from the `employee` table. Each time the loop iterates, the OUT parameter *counter* is incremented and the value of *v_midinit* is checked to ensure that the value is not a single space (' '). If *v_midinit* is a single space, the LEAVE statement passes the flow of control outside of the loop.

```
CREATE PROCEDURE LOOP_UNTIL_SPACE(OUT counter INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE v_firstnme VARCHAR(12);
    DECLARE v_midinit CHAR(1);
    DECLARE v_lastname VARCHAR(15);
    DECLARE c1 CURSOR FOR
      SELECT firstnme, midinit, lastname
        FROM employee;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET counter = -1;
    OPEN c1;
    fetch_loop:
    LOOP
      FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
      IF v_midinit = ' ' THEN
        LEAVE fetch_loop;
      END IF;
      SET v_counter = v_counter + 1;
    END LOOP fetch_loop;
    SET counter = v_counter;
    CLOSE c1;
  END
```

MERGE

The MERGE statement updates a target (a table or view, or the underlying tables or views of a fullselect) using data from a source (result of a table reference).

Rows in the target that match the source can be deleted or updated as specified, and rows that do not exist in the target can be inserted. Updating, deleting or inserting a row in a view updates, deletes or inserts the row in the tables on which the view is based.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- If an insert operation is specified, INSERT privilege on the table or view; if a delete operation is specified, DELETE privilege on the table or view; and if an update operation is specified, either:
  - UPDATE privilege on the table or view
  - UPDATE privilege on each column that is to be updated
- CONTROL privilege on the table
- DATAACCESS authority

The privileges held by the authorization ID of the statement must also include at least one of the following authorities:

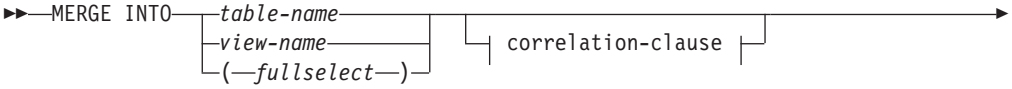
- SELECT privilege on every table or view identified in the *table-reference*
- CONTROL privilege on the tables or views identified in the *table-reference*
- DATAACCESS authority

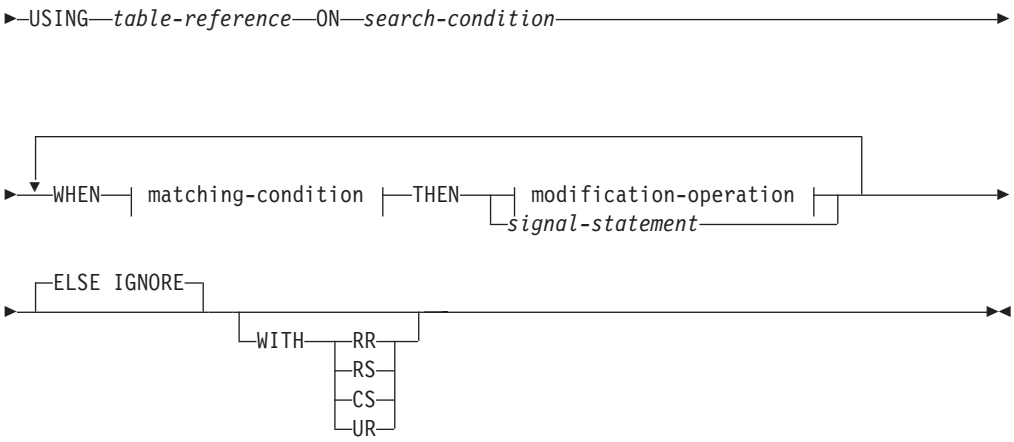
If *search-condition*, *insert-operation*, or *assignment-clause* includes a subquery, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

- SELECT privilege on every table or view identified in the subquery
- CONTROL privilege on the tables or views identified in the subquery
- DATAACCESS authority

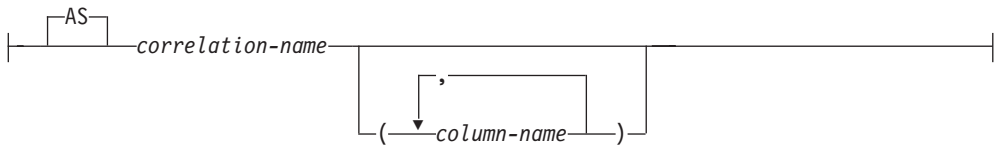
If an expression that refers to a function is specified, the privilege set must include any authority that is necessary to execute the function.

Syntax

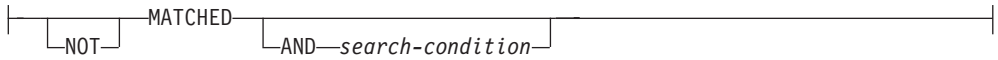




correlation-clause:

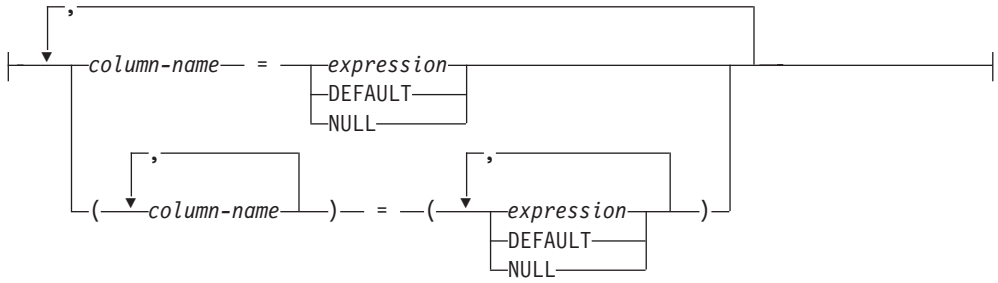


matching-condition:

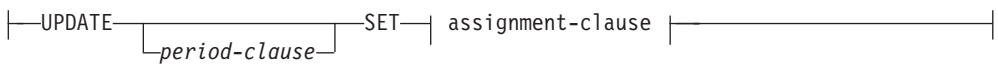


modification-operation:

assignment-clause:

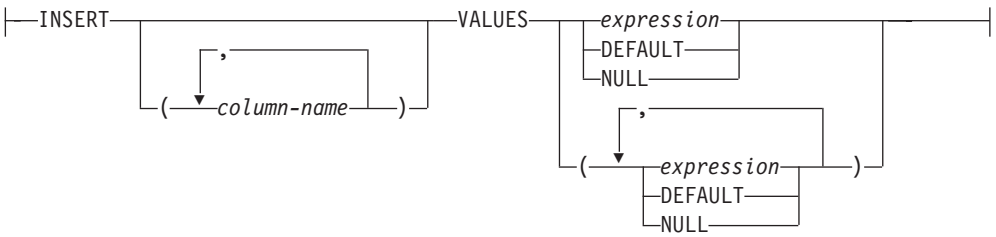


update-operation:



delete-operation:

insert-operation:



period-clause:

Description

table-name, *view-name*, **or** *(fullselect)*
Identifies the target of the update, delete, or insert operations of the merge. The name must identify a table or view that exists at the current server, but it must not identify a catalog table, a system-maintained materialized query table, a view of a catalog table, a read-only view, or a view that directly or indirectly contains a WHERE clause that references a subquery or a routine defined with NOT DETERMINISTIC or EXTERNAL ACTION (SQLSTATE 42807).

If the target of the merge operation is a fullselect, the fullselect must be updatable, deletable, or insertable as defined in the "Updatable views", "Deletable views", or "Insertable views" Notes items in the description of the CREATE VIEW statement.

You cannot use a *period-clause* in an update-operation or a delete-operation if the target of the merge operation is a union-all view or a fullselect.

You cannot use a nickname (a reference to a remote, federated table) as the target table.

correlation-clause
Can be used within *search-condition* or on the right side of an *assignment-clause* to designate a table, view, or fullselect. For a description of *correlation-clause*, see "table-reference" in the description of "Subselect".

USING *table-reference*
Specifies a set of rows as a result table to be merged into the target. If the result table is empty, a warning is returned (SQLSTATE 02000).

ON *search-condition*
Specifies which rows from *table-reference* are to be used in the update and delete operation of the merge, and which rows are to be used in the insert operation of the merge. The *search-condition* is applied to each row of the target table and result table of the *table-reference*. For those rows of the result table of the *table-reference* where the result of the *search-condition* is true, the specified

update or delete operation is performed. For those rows of the result table of the *table-reference* where the result of the *search-condition* is not true, the specified insert operation is performed.

The *search-condition* has the following restrictions (SQLSTATE 42972 unless otherwise noted):

- It cannot contain any subqueries, scalar or otherwise
- It cannot include any dereference operations or the DEREF function where the reference value is other than the object identifier column
- It cannot include an SQL function
- It cannot include an XMLQUERY or XMLEXISTS expression
- Any column that is referenced in an expression of the *search-condition* must be a column of the target table, view, or *table-reference*
- Any function that is referenced in an expression of the *join-condition* of a full outer join must be deterministic and have no external action
- It cannot be include an aggregate function (SQLSTATE 42903)

If the *search-condition* is false or unknown for every row in *table-reference*, a warning is returned (SQLSTATE 02000).

WHEN *matching-condition*
Specifies the condition under which the *modification-operation* or the *signal-statement* is executed. Each *matching-condition* is evaluated in order of specification. Rows for which the *matching-condition* evaluates to true are not considered in subsequent matching conditions.

MATCHED
Indicates the operation to be performed on the rows where the ON search condition is true. Only UPDATE, DELETE, or *signal-statement* can be specified after THEN.

AND *search-condition*
Specifies a further search condition to be applied against the rows that matched the ON search condition for the operation to be performed after THEN.

NOT MATCHED
Indicates the operation to be performed on the rows where the ON search condition is false or unknown. Only INSERT or *signal-statement* can be specified after THEN.

AND *search-condition*
Specifies a further search condition to be applied against the rows that did not match the ON search condition for the operation to be performed after THEN.

THEN *modification-operation*
Specifies the operation to execute when the *matching-condition* evaluates to true.

update-operation
Specifies the update operation to be executed for the rows where the *matching-condition* evaluates to true.

UPDATE
Introduces the update operation.

period-clause
Specifies that a period clause is applied to the update operation in

the MERGE statement. For more information about the effects of a period clause specified in the context of an update operation, see the UPDATE statement topic.

SET
Introduces the assignment of values to column names.

assignment-clause
Specifies a list of column updates.

column-name
Identifies a column to be updated. The *column-name* must identify a column of the specified table or view, but not a view column derived from a scalar function, constant, or expression. A column must not be specified more than once (SQLSTATE 42701).

A view column derived from the same column as another column of the view can be updated, but both columns cannot be updated in the same MERGE statement (SQLSTATE 42701).

expression
Indicates the new value of the column. The *expression* must not include an aggregate function except when it occurs within a scalar fullselect (SQLSTATE 42903).

An *expression* can contain references to columns of the *table-name* or *view-name*. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated.

If *expression* is a reference to a single column of the source table, the source table column value may have been specified with an extended indicator variable value. The effects of such indicator variables apply to the corresponding target columns of the *assignment-clause*.

If *expression* is a single host variable, or a host variable being explicitly cast, the host variable can include an indicator variable that is enabled for extended indicator variables.

When extended indicator variables are enabled, the extended indicator variable values of default (-5) or unassigned (-7) must not be used (SQLSTATE 22539) if either of the following statements is true:

- The expression is more complex than a single host variable with explicit casts
- The target column has data type of structured type

DEFAULT
The default value assigned to the column. DEFAULT can be specified only for columns that have a default value. For information about default values of data types, see the description of the DEFAULT clause in the “CREATE TABLE” statement.

DEFAULT must be specified for a column that was defined as GENERATED ALWAYS. A valid value can be specified for a column that was defined as GENERATED BY DEFAULT.

NULL
Specifies the null value as the new value of the column. Specify NULL only for nullable columns (SQLSTATE 23502).

delete-operation
Specifies the delete operation to be executed for the rows where the *matching-condition* evaluates to true.

DELETE
Introduces the delete operation.

period-clause
Specifies that a period clause is applied to the delete operation in the MERGE statement. For more information about the effects of a period clause specified in the context of a delete operation, see the DELETE statement topic.

insert-operation
Specifies the insert operation to be executed for the rows where the *matching-condition* evaluates to true.

INSERT
Introduces a list of column names and row value expressions to be used for the insert operation.

The number of values for the row in the row value expression must equal the number of names in the insert column list. The first value is inserted in the first column in the list, the second value in the second column, and so on.

(*column-name***,...)**
Specifies the columns for which the insert values are provided. Each name must identify a column of the table or view. The same column must not be identified more than once (SQLSTATE 42701). A view column that cannot accept insert values must not be identified. A value cannot be inserted into a view column that is derived from:

- A constant, expression, or scalar function
- The same base table column as some other column of the view

If the object of the operation is a view with such columns, a list of column names must be specified, and the list must not identify these columns.

Omission of the column list is an implicit specification of a list in which every column of the table (that is not defined as implicitly hidden) or view is identified in left-to-right order. This list is established when the statement is prepared, and therefore does not include columns that were added to a table after the statement was prepared.

VALUES
Introduces one or more rows of values to be inserted.

expression
Any expression that does not include a column name (SQLSTATE 42703).

If *expression* is a reference to a single column of the source table, the source table column value may have been specified with an extended indicator variable value. The effects of such indicator variables apply to the corresponding target columns of the *insert-operation*.

If *expression* is a single host variable, or a host variable being explicitly cast, the host variable can include an indicator variable (or in the case of a host structure, an indicator array) that is enabled for extended indicator variables.

When extended indicator variables are enabled, the extended indicator variable values of default (-5) or unassigned (-7) must not be used (SQLSTATE 22539) if either of the following statements is true:

- The expression is more complex than a single host variable with explicit casts
- The target column has data type of structured type

DEFAULT
The default value assigned to the column. DEFAULT can be specified only for columns that have a default value. For information about default values of data types, see the description of the DEFAULT clause in the "CREATE TABLE" statement.

DEFAULT must be specified for a column that was defined as GENERATED ALWAYS. A valid value can be specified for a column that was defined as GENERATED BY DEFAULT.

NULL
Specifies the null value as the value of the column. Specify NULL only for nullable columns (SQLSTATE 23502).

signal-statement
Specifies the SIGNAL statement that is to be executed to return an error when the *matching-condition* evaluates to true.

ELSE IGNORE
Specifies that no action is to be taken for the rows where no *matching-condition* evaluates to true. If all rows of *table-reference* are ignored, a warning is returned (SQLSTATE 02000).

WITH
Specifies the isolation level at which the MERGE statement is executed.

RR Repeatable Read

RS Read Stability

CS Cursor Stability

UR Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound.

Rules

- More than one *modification-operation* (UPDATE SET, DELETE, or *insert-operation*), or *signal-statement* can be specified in a single MERGE statement.
- Each row in the target can only be operated on once. A row in the target can only be identified as MATCHED with one row in the result table of the *table-reference* (SQLSTATE 21506). A nested SQL operation (RI or trigger except INSTEAD OF trigger) cannot specify the target table (or a table within the same table hierarchy) as a target of an UPDATE, DELETE, INSERT, or MERGE statement (SQLSTATE 27000).
- ***Security policy:*** If the identified target table or the base table of the identified target view is protected with a security policy, the session authorization ID must have the label-based access control (LBAC) credentials that allow the following types of access.
  - For the update operation:
    - Write access to all protected columns that are being updated (SQLSTATE 42512)
    - Write access for any explicit value provided for a DB2SECURITYLABEL column for security policies that were created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option (SQLSTATE 23523)
    - Read and write access to all rows that are being updated (SQLSTATE 42519)

    The session authorization ID must also have been granted a security label for write access for the security policy if an implicit value is used for a DB2SECURITYLABEL column (SQLSTATE 23523), which can happen when:
    - The DB2SECURITYLABEL column is not included in the list of columns that are to be updated (and so it will be implicitly updated to the security label for write access of the session authorization ID)
    - A value for the DB2SECURITYLABEL column is explicitly provided but the session authorization ID does not have write access for that value, and the security policy is created with the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option
  - For the delete operation:
    - Write access to all protected columns (SQLSTATE 42512)
    - Read and write access to all of the rows that are selected for deletion (SQLSTATE 42519)
  - For the insert operation:
    - Write access to all protected columns for which a data value is explicitly provided (SQLSTATE 42512)
    - Write access for any explicit value provided for a DB2SECURITYLABEL column for security policies that were created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option (SQLSTATE 23523)

    The session authorization ID must also have been granted a security label for write access for the security policy if an implicit value is used for a DB2SECURITYLABEL column (SQLSTATE 23523), which can happen when:
    - A value for the DB2SECURITYLABEL column is not explicitly provided
    - A value for the DB2SECURITYLABEL column is explicitly provided but the session authorization ID does not have write access for that value, and the security policy is created with the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option

- ***INSTEAD OF triggers***: If a view is specified as the target of the MERGE statement, either no INSTEAD OF triggers should be defined for the view, or an INSTEAD OF trigger should be defined for each of the update, delete, and insert operations (SQLSTATE 428FZ).
- **Extended indicator variable usage**: If enabled, negative indicator variable values outside the range of -1 through -7 must not be input (SQLSTATE 22010). Also, if enabled, the default and unassigned extended indicator variable values must not appear in contexts in which they are not supported (SQLSTATE 22539).
- **Extended indicator variables in the *assignment-clause***: An *expression* that is a reference to a single column of the source table, a single host variable, or a host variable being explicitly cast can result in assigning an extended indicator variable-based value. Assigning the extended indicator variable-based value of unassigned has the effect of leaving the target column set to its current value, as if it had not been specified in the statement. Assigning the extended indicator variable-based value of default assigns the default value of the column. For information on default values of data types, see the description of the DEFAULT clause in "CREATE TABLE" on page 672.

  If a target column is not updatable (for example, a column in a view that is defined as an expression), then it must be assigned the extended indicator variable-based value of unassigned (SQLSTATE 42808).

  If the target column is a column defined as GENERATED ALWAYS, then it must be assigned the DEFAULT keyword, or the extended indicator variable-based values of default or unassigned (SQLSTATE 428C9).

  The *assignment-clause* must not assign all target columns to an extended indicator variable-based value of unassigned (SQLSTATE 22540).
- **Extended indicator variables in the** *insert-operation*: An *expression* that is a reference to a single column of the source table, a single host variable, or a host variable being explicitly cast can result in inserting an extended indicator variable-based value. In *insert-operation*, a value of unassigned has the effect of setting the column to its default value.

  If a target column is not updatable, then it must be assigned the extended indicator variable-based value of unassigned (SQLSTATE 42808), unless it is a column defined as GENERATED ALWAYS. If the target column is a column defined as GENERATED ALWAYS, then it must be assigned the DEFAULT keyword, or the extended indicator variable-based values of default or unassigned (SQLSTATE 428C9).

For other rules that affect the update, insert, or delete operation portion of the MERGE statement, see the "Rules" section of the corresponding statement description.

Notes

- ***Order of processing***:
  1. Determine the set of rows to be processed from the source and target. If CURRENT TIMESTAMP is used in this statement, only one clock reading is done for the whole statement.
  2. Use the ON clause to classify these rows as either MATCHED or NOT MATCHED.
  3. Evaluate any *matching-condition* in the WHEN clauses.
  4. Evaluate any *expression* in any *assignment-clause* and *insert-operation*.
  5. Execute each *signal-statement*.

6. Apply each *modification-operation* to the applicable rows in the order of specification. The constraints and triggers activated by each *modification-operation* are executed for the *modification-operation*. Statement-level triggers are activated even if no rows satisfy the *modification-operation*. Each *modification-operation* can affect the triggers and referential constraints of each subsequent *modification-operation*.

- ***Statement level atomicity***: If an error occurs during execution of the MERGE statement, the whole statement is rolled back.
- ***Number of rows updated***: When a MERGE statement completes execution, the value of the ROW_COUNT item for GET DIAGNOSTICS and SQLERRD(3) in the SQLCA is the number of rows operated on by the MERGE statement, excluding rows identified by the ELSE IGNORE clause. The value in SQLERRD(3) does not include the number of rows that were operated on as a result of constraints or triggers. The value in SQLERRD(5) includes the number of these rows.
- ***Inserted row cannot also be updated***: No attempt is made to update a row in the target that did not already exist before the MERGE statement was executed; that is, there are no updates of rows that were inserted by the MERGE statement.
- **Extended indicator variables and update triggers**: If a target column has been assigned with an extended indicator variable-based value of unassigned, that column is not considered to have been updated. That column is treated as if it had not been specified in the OF *column-name* list of any update trigger defined on the target table.
- **Extended indicator variables and insert triggers**: No change in the activation of insert triggers results from the use of extended indicator variables. If all columns in the implicit or explicit column list have been assigned to an extended indicator variable-based value of unassigned or default, an insert where all columns have their respective default values is attempted. If the insert is successful, the insert trigger is activated.
- **Extended indicator variables and deferred error checks**: When extended indicator variables are enabled, validation that would otherwise be done in statement preparation to recognize an insert into, or update of, a non-updatable column, is deferred until statement execution. Whether an error should be reported can be determined only during execution.
- **Considerations for system-period temporal tables**: When MERGE is processed for a system-period temporal table, the rows are impacted in the same way as if the specific data change operations had been invoked. See UPDATE statement, DELETE statement, and INSERT statement topics for more information.
- **Considerations for application-period temporal tables and triggers**; When a row is deleted and the FOR PORTION OF BUSINESS_TIME clause is specified, additional rows may be implicitly inserted to reflect any portion of the row that was not deleted. Any existing delete triggers are activated for the rows deleted, and any existing insert triggers are activated for rows that are implicitly inserted. When a row is updated and the FOR PORTION OF BUSINESS_TIME clause is specified, additional rows may be implicitly inserted to reflect any portion of the row that was not updated. Any existing update triggers are activated for the rows updated, and any existing insert triggers are activated for rows that are implicitly inserted.
- **Considerations for a MERGE without a column list in the insert-operation**: A MERGE statement without a column list specified as part of the insert-operation does not include implicitly hidden columns. Columns that are defined as implicitly hidden and not null must have a defined default value.

Examples

- *Example 1:* For activities whose description has been changed, update the description in the archive table. For new activities, insert into the archive table. The archive and activities table both have activity as a primary key.

```
MERGE INTO archive ar
USING (SELECT activity, description FROM activities) ac
ON (ar.activity = ac.activity)
WHEN MATCHED THEN
   UPDATE SET
      description = ac.description
WHEN NOT MATCHED THEN
   INSERT
      (activity, description)
      VALUES (ac.activity, ac.description)
```

- *Example 2:* Using the shipment table, merge rows into the inventory table, increasing the quantity by part count in the shipment table for rows that match; else insert the new *partno* into the inventory table.

```
MERGE INTO inventory AS in
USING (SELECT partno, description, count FROM shipment
       WHERE shipment.partno IS NOT NULL) AS sh
ON (in.partno = sh.partno)
WHEN MATCHED THEN
   UPDATE SET
      description = sh.description,
      quantity = in.quantity + sh.count
WHEN NOT MATCHED THEN
   INSERT
      (partno, description, quantity)
      VALUES (sh.partno, sh.description, sh.count)
```

- *Example 3:* Using the transaction table, merge rows into the account table, updating the balance from the set of transactions against an account ID and inserting new accounts from the consolidated transactions where they do not already exist.

```
MERGE INTO account AS a
USING (SELECT id, sum(amount) sum_amount FROM transaction
       GROUP BY id) AS t
ON a.id = t.id
WHEN MATCHED THEN
   UPDATE SET
      balance = a.balance + t.sum_amount
WHEN NOT MATCHED THEN
   INSERT
      (id, balance)
      VALUES (t.id, t.sum_amount)
```

- *Example 4:* Using the transaction_log table, merge rows into the employee_file table, updating the phone and office with the latest transaction_log row based on the transaction time, and inserting the latest new employee_file row where the row does not already exist.

```
MERGE INTO employee_file AS e
USING (SELECT empid, phone, office
       FROM (SELECT empid, phone, office,
             ROW_NUMBER() OVER (PARTITION BY empid
             ORDER BY transaction_time DESC) rn
             FROM transaction_log) AS nt
             WHERE rn = 1) AS t
ON e.empid = t.empid
WHEN MATCHED THEN
   UPDATE SET
      (phone, office) =
      (t.phone, t.office)
WHEN NOT MATCHED THEN
```

```
   INSERT
      (empid, phone, office)
      VALUES (t.empid, t.phone, t.office)
```

- *Example 5:* Using dynamically supplied values for an employee row, update the master employee table if the data corresponds to an existing employee, or insert the row if the data is for a new employee. The following example is a fragment of code from a C program.

```
hv1 =
"MERGE INTO employee AS t
USING TABLE(VALUES(CAST (? AS CHAR(6)), CAST (? AS VARCHAR(12)),
                   CAST (? AS CHAR(1)), CAST (? AS VARCHAR(15)),
                   CAST (? AS SMALLINT), CAST (? AS INTEGER)))
                   s(empno, firstnme, midinit, lastname, edlevel, salary)
ON t.empno = s.empno
WHEN MATCHED THEN
   UPDATE SET
      salary = s.salary
WHEN NOT MATCHED THEN
   INSERT
      (empno, firstnme, midinit, lastname, edlevel, salary)
      VALUES (s.empno, s.firstnme, s.midinit, s.lastname, s.edlevel,
              s.salary)";
EXEC SQL PREPARE s1 FROM :hv1;
EXEC SQL EXECUTE s1 USING '000420', 'SERGE', 'K', 'FIELDING', 18, 39580;
```

- *Example 6:* Update the list of activities organised by Group A in the archive table. Delete all outdated activities and update the activities information (description and date) in the archive table if they have been changed. For new upcoming activities, insert into the archive. Signal an error if the date of the activity is not known. The date of the activities in the archive table must be specified. Each group has an activities table. For example, activities_groupA contains all activities that they organize, and the archive table contains all upcoming activities organized by different groups in a company. The archive table has (group, activity) as the primary key, and date is not nullable. All activities tables have activity as the primary key. The last_modified column in the archive is defined with CURRENT TIMESTAMP as the default value.

```
MERGE INTO archive ar
USING (SELECT activity, description, date, last_modified
       FROM activities_groupA) ac
ON (ar.activity = ac.activity) AND ar.group = 'A'
WHEN MATCHED AND ac.date IS NULL THEN
   SIGNAL SQLSTATE '70001'
      SET MESSAGE_TEXT =
         ac.activity CONCAT ' cannot be modified. Reason: Date is not known'
WHEN MATCHED AND ac.date < CURRENT DATE THEN
   DELETE
WHEN MATCHED AND ar.last_modified < ac.last_modified THEN
   UPDATE SET
      (description, date, last_modified) = (ac.description, ac.date, DEFAULT)
WHEN NOT MATCHED AND ac.date IS NULL THEN
   SIGNAL SQLSTATE '70002'
      SET MESSAGE_TEXT =
         ac.activity CONCAT ' cannot be inserted. Reason: Date is not known'
WHEN NOT MATCHED AND ac.date >= CURRENT DATE THEN
   INSERT
      (group, activity, description, date)
      VALUES ('A', ac.activity, ac.description, ac.date)
ELSE IGNORE
```

OPEN

The OPEN statement opens a cursor so that it can be used to fetch rows from its result table.

Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared. When invoked using the command line processor, some options cannot be specified. For more information, refer to "Using command line SQL statements and XQuery statements".

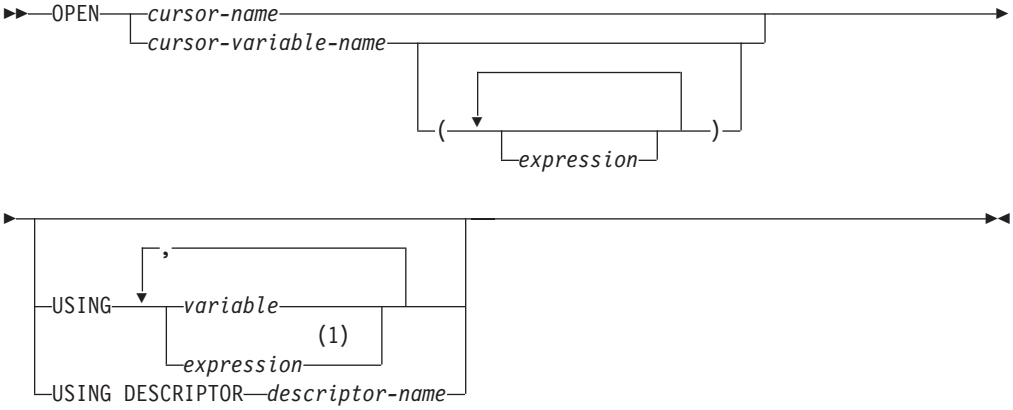
Authorization

If a global variable is referenced, the privileges held by the authorization ID of the statement must include one of the following authorities:

- READ privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

Group privileges are not considered because this statement cannot be dynamically prepared.

Syntax



Notes:

1 An expression other than a variable can only be used in compiled compound statements.

Description

cursor-name

Names a cursor that is defined in a DECLARE CURSOR statement that was stated earlier in the program. If *cursor-name* identifies a cursor in an SQL procedure declared as WITH RETURN TO CLIENT that is already in the open state, the existing open cursor becomes a result set cursor that is no longer accessible using cursor-name and a new cursor is opened that becomes accessible using cursor-name. Otherwise, when the OPEN statement is executed, the cursor identified by *cursor-name* must be in the closed state.

The DECLARE CURSOR statement must identify a SELECT statement, in one of the following ways:

- Including the SELECT statement in the DECLARE CURSOR statement
- Including a *statement-name* that names a prepared SELECT statement.

The result table of the cursor is derived by evaluating the SELECT statement. The evaluation uses the current values of any special registers, global variables, or PREVIOUS VALUE expressions specified in the SELECT statement, and the current values of any host variables specified in the SELECT statement or the USING clause of the OPEN statement. The rows of the result table may be derived during the execution of the OPEN statement, and a temporary table may be created to hold them; or they may be derived during the execution of subsequent FETCH statements. In either case, the cursor is placed in the open state and positioned before the first row of its result table. If the table is empty, the state of the cursor is effectively "after the last row".

cursor-variable-name

Names a cursor variable. The value of the cursor variable must not be null (SQLSTATE 34000). A cursor variable that is directly or indirectly assigned a cursor value constructor can be used only in an OPEN statement that is in the same scope as the assignment (SQLSTATE 51044). If the cursor value constructor assigned to the cursor variable specified a *statement-name*, the OPEN statement must be in the same scope where that *statement-name* was explicitly or implicitly declared (SQLSTATE 51044).

When the OPEN statement is executed, the underlying cursor of the cursor variable must be in the closed state. The result table of the underlying cursor is derived by evaluating the SELECT statement or dynamic statement associated with the cursor variable. The evaluation uses the current values of any special registers, global variables, or PREVIOUS VALUE expressions specified in the SELECT statement, and the current values of any variables specified in the SELECT statement or the USING clause of the OPEN statement. The rows of the result table may be derived during the execution of the OPEN statement, and a temporary table may be created to hold them; or they may be derived during the execution of subsequent FETCH statements. In either case, the cursor is placed in the open state and positioned before the first row of its result table. If the table is empty, the state of the cursor is effectively "after the last row".

An OPEN statement using a *cursor-variable-name* can only be used within a compound SQL (compiled) statement.

(expression, ...)

Specifies the arguments associated with the named parameters of a parameterized cursor variable. The *cursor-value-constructor* assigned to the cursor variable must include a list of parameters with the same number of parameters as the number of arguments specified (SQLSTATE 07006 or 07004). The data type and value of the *n*th expression must be assignable to the *n*th parameter (SQLSTATE 07006 or 22018).

USING

Introduces the values that are substituted for the parameter markers or variables in the statement of the cursor. For an explanation of parameter markers, see “PREPARE”.

If a *statement-name* is specified in the DECLARE CURSOR statement or the cursor value constructor associated with the cursor variable that includes

parameter markers, USING must be used. If the prepared statement does not include parameter markers, USING is ignored.

If a *select-statement* is specified in the DECLARE CURSOR statement or the non-parameterized cursor value constructor associated with the cursor variable, USING may be used to override the variable values.

variable

Identifies a variable or a host structure declared in the program in accordance with the rules for declaring variables and host variables. The number of variables must be the same as the number of parameter markers in the prepared statement. The *n*th variable corresponds to the *n*th parameter marker in the prepared statement. Where appropriate, locator variables and file reference variables can be provided as the source of values for parameter markers.

expression

Specifies values to associate with parameter markers using expressions. An OPEN statement that specifies expressions in the USING clause can only be used within a compound SQL (compiled) statement (SQLSTATE 42601). The number of expressions must be the same as the number of parameter markers in the prepared statement (SQLSTATE 07001). The *n*th expression corresponds to the *n*th parameter marker in the prepared statement. The data type and value of the nth expression must be assignable to the type associated with the nth parameter marker (SQLSTATE 07006).

Rules

- When the SELECT statement of the cursor is evaluated, each parameter marker in the statement is effectively replaced by its corresponding host variable. For a typed parameter marker, the attributes of the target variable are those specified by the CAST specification. For an untyped parameter marker, the attributes of the target variable are determined according to the context of the parameter marker.
- Let V denote a host variable that corresponds to parameter marker P. The value of V is assigned to the target variable for P in accordance with the rules for assigning a value to a column. Thus:
  - V must be compatible with the target.
  - If V is a string, its length (excluding trailing blanks for strings that are not long strings) must not be greater than the length attribute of the target.
  - If V is a number, the absolute value of its integral part must not be greater than the maximum absolute value of the integral part of the target.
  - If the attributes of V are not identical to the attributes of the target, the value is converted to conform to the attributes of the target.

  When the SELECT statement of the cursor is evaluated, the value used in place of P is the value of the target variable for P. For example, if V is CHAR(6), and the target is CHAR(8), the value used in place of P is the value of V padded with two blanks.
- The USING clause is intended for a prepared SELECT statement that contains parameter markers. However, it can also be used when the SELECT statement of the cursor is part of the DECLARE CURSOR statement or the non-parameterized cursor value constructor associated with the cursor variable. In this case the OPEN statement is executed as if each host variable in the SELECT statement were a parameter marker, except that the attributes of the target variables are the same as the attributes of the host variables in the SELECT statement. The effect is to override the values of the host variables in the SELECT statement of the

cursor with the values of the host variables specified in the USING clause. A variable value override must not be used when opening a parameterized cursor variable since the SELECT statement will not include any other variables.

- SQL data change statements and routines that modify SQL data embedded in the cursor definition are completely executed, and the result set is stored in a temporary table when the cursor opens. If statement execution is successful, the SQLERRD(3) field contains the sum of the number of rows that qualified for insert, update, and delete operations. If an error occurs during execution of an OPEN statement involving a cursor that contains a data change statement within a fullselect, the results of that data change statement are rolled back.

  Explicit rollback of an OPEN statement, or rollback to a savepoint before an OPEN statement, closes the cursor. If the cursor definition contains a data change statement within the FROM clause of a fullselect, the results of the data change statement are rolled back.

  Changes to rows in a table that is targeted by a data change statement nested within a SELECT statement or a SELECT INTO statement are processed when the cursor opens, and are not undone if an error occurs during a fetch operation against that cursor.

Notes

- ***Closed state of cursors:*** All cursors in a program are in the closed state when the program is initiated and when it initiates a ROLLBACK statement.

  All cursors, except open cursors declared WITH HOLD, are in a closed state when a program issues a COMMIT statement.

  A cursor can also be in the closed state because a CLOSE statement was executed or an error was detected that made the position of the cursor unpredictable.

  The underlying cursor of a cursor variable is closed if the cursor variable goes out of scope and there are no other cursor variables that referenced that underlying cursor.
- To retrieve rows from the result table of a cursor, execute a FETCH statement when the cursor is open. The only way to change the state of a cursor from closed to open is to execute an OPEN statement.
- ***Effect of materalized result tables:*** In some cases, such as when the cursor is not read only, the result rows of a cursor are derived during the execution of FETCH statements. In other cases, the materialized result table method is used instead. With the materialized result table method the entire result table is transferred to a temporary buffer during the execution of the OPEN statement. When a temporary buffer is used, the results of a program can differ in these ways:
  - An error can occur during OPEN that would otherwise not occur until some later FETCH statement.
  - INSERT, UPDATE, and DELETE statements executed in the same transaction while the cursor is open cannot affect the result table.
  - Any NEXT VALUE expressions in the SELECT statement are evaluated for every row of the result table during OPEN.

  Conversely, if a temporary buffer is not used, INSERT, UPDATE, and DELETE statements executed while the cursor is open can affect the result table if issued from the same unit of work, and any NEXT VALUE expressions in the SELECT statement are evaluated as each row is fetched. This result table can also be affected by operations executed by the same unit of work, and the effect of such operations is not always predictable. For example, if cursor C is positioned on a row of its result table defined as SELECT * FROM T, and a new row is inserted

into T, the effect of that insert on the result table is not predictable because its rows are not ordered. Thus a subsequent FETCH C may or may not retrieve the new row of T.

- Statement caching affects cursors declared open by the OPEN statement.
- ***Opening the same cursor multiple times:*** A cursor in an SQL procedure declared as WITH RETURN TO CLIENT can be opened even when a cursor with the same name is already in the open state. In this case, the existing open cursor becomes a result set cursor and is no longer accessible by its cursor name. A new cursor is opened and becomes accessible by the cursor name. Closing the new cursor does not make the cursor that was previously accessible by that name accessible by the cursor name again. The cursors that become result set cursors in this way cannot be accessed at the server and can be processed only at the client.

Examples

Example 1: Write the embedded statements in a COBOL program that will:

1. Define a cursor C1 that is to be used to retrieve all rows from the DEPARTMENT table for departments that are administered by (ADMRDEPT) department 'A00'.
2. Place the cursor C1 before the first row to be fetched.

```
EXEC SQL  DECLARE C1 CURSOR FOR
               SELECT DEPTNO, DEPTNAME, MGRNO
                 FROM DEPARTMENT
                 WHERE ADMRDEPT = 'A00'
END-EXEC.


EXEC SQL  OPEN C1
END-EXEC.
```

Example 2: Code an OPEN statement to associate a cursor DYN_CURSOR with a dynamically defined select-statement in a C program. Assuming two parameter markers are used in the predicate of the select-statement, two host variable references are supplied with the OPEN statement to pass integer and varchar(64) values between the application and the database. (The related host variable definitions, PREPARE statement, and DECLARE CURSOR statement are also shown in this example.)

```
EXEC SQL  BEGIN DECLARE SECTION;
  static short    hv_int;
  char            hv_vchar64[65];
  char            stmt1_str[200];
EXEC SQL  END DECLARE SECTION;

EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

EXEC SQL  OPEN DYN_CURSOR USING :hv_int, :hv_vchar64;
```

Example 3: Code an OPEN statement as in example 2, but in this case the number and data types of the parameter markers in the WHERE clause are not known.

```
EXEC SQL  BEGIN DECLARE SECTION;
  char    stmt1_str[200];
EXEC SQL  END DECLARE SECTION;
EXEC SQL  INCLUDE SQLDA;

EXEC SQL  PREPARE STMT1_NAME FROM :stmt1_str;
```

```
EXEC SQL  DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

EXEC SQL  OPEN DYN_CURSOR USING DESCRIPTOR :sqlda;
```

Example 4: Create a procedure that does the following operations:

1. Assigns a cursor to the output cursor variable
2. Opens the cursor

```
CREATE PROCEDURE PROC1 (OUT P1 CURSOR)LANGUAGE SQL
BEGIN
SET P1=CURSOR FOR SELECT DEPTNO, DEPTNAME, MGRNO FROM DEPARTMENT WHERE ADMRDEPT='A00'; --
OPEN P1; --
END;
```

PIPE

The PIPE statement is used to return a row from a compiled table function.

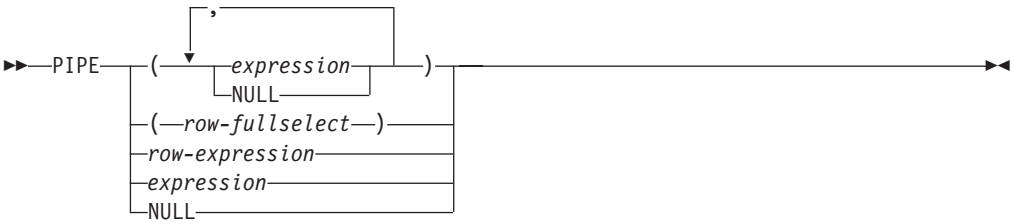
Invocation

This statement can be embedded in a compound SQL (compiled) statement of an SQL table function. It is not an executable statement and cannot be dynamically prepared.

Authorization

No privileges are required to invoke the PIPE statement. However, the authorization ID of the statement must hold the necessary privileges to invoke any expression that is embedded in the PIPE statement.

Syntax



Description

(*expression*, ...)
Specifies a row value is returned from the function. The number of expressions (or NULL keywords) in the list must match the RETURNS data type of the function and the value of each expression must be assignable to the corresponding column or field in the RETURNS data type of the function.

row-fullselect
Specifies a fullselect that returns a single row with the number of columns corresponding to the number of columns or fields in the RETURNS data type of the function. The value in each column of the row returned by the fullselect must be assignable to the corresponding column or field in the RETURNS data type of the function. If the result of the row fullselect is no rows, null values are returned.

row-expression
Specifies the row value is returned from the function. The number of fields in the row must match the RETURNS data type of the function and each field in the row must be assignable to the corresponding field in the RETURNS data type of the function. If the *row-expression* and the RETURNS data type are user-defined row types, the type names must be the same (SQLSTATE 42821).

expression
Specifies a scalar value is returned from the function. The RETURNS data type of the table function must have a single column and the expression value must be assignable to that column.

NULL
Specifies that a null value is returned from the function. A null value is returned for each column or row field.

Notes

- **Locally declared procedures:** The PIPE statement cannot be used within a procedure that is locally declared in the compound SQL (compiled) statement of an SQL table function.
- **Similar terms:** An SQL table function that uses a PIPE statement is sometimes referred to as a *pipelined* function.

PREPARE

The PREPARE statement is used by application programs to dynamically prepare an SQL statement for execution. The PREPARE statement creates an executable SQL statement, called a *prepared statement*, from a character string form of the statement, called a *statement string*.

Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

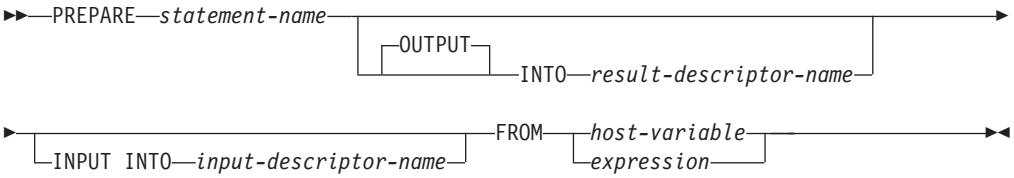
For statements where authorization checking is performed at statement preparation time (DML), the privileges held by the authorization ID of the statement must include those required to execute the SQL statement specified by the PREPARE statement. The authorization ID of the statement might be affected by the DYNAMICRULES bind option.

For statements where authorization checking is performed at statement execution time (DDL, GRANT, and REVOKE statements), no authorization is required to use this statement; however, the authorization is checked when the prepared statement is executed.

For statements involving tables that are protected with a security policy, the rules associated with the security policy are always evaluated at statement execution time.

If the authorization ID of the statement holds EXPLAIN, SQLADM, or DBADM authority, the user may prepare any statement; however, the ability to execute the statement is re-checked at statement execution time.

Syntax



Description

statement-name
Names the prepared statement. If the name identifies an existing prepared statement, that previously prepared statement is destroyed. The name must not identify a prepared statement that is the SELECT statement of an open cursor.

OUTPUT INTO
If OUTPUT INTO is used, and the PREPARE statement executes successfully, information about the output parameter markers in the prepared statement is placed in the SQLDA specified by *result-descriptor-name*.

result-descriptor-name
Specifies the name of an SQLDA. (The DESCRIBE statement may be used as an alternative to this clause.)

INPUT INTO
If INPUT INTO is used, and the PREPARE statement executes successfully, information about the input parameter markers in the prepared statement is placed in the SQLDA specified by *input-descriptor-name*. Input parameter markers are always considered nullable, regardless of usage.

input-descriptor-name
Specifies the name of an SQLDA. (The DESCRIBE statement may be used as an alternative to this clause.)

FROM
Introduces the statement string. The statement string is the value of the specified host variable.

host-variable
Specifies a host variable that is described in the program in accordance with the rules for declaring character string variables. It must be a fixed-length or varying-length character-string variable that is less than the maximum statement size of 2 097 152 bytes. Note that a CLOB(2097152) can contain a maximum size statement, but a VARCHAR cannot.

expression
An expression specifying the statement string. The expression must return a fixed-length or varying-length character-string type that is less than the maximum statement size of 2 097 152 bytes.

Rules

- ***Rules for statement strings:*** The statement string must be an executable statement that can be dynamically prepared. It must be one of the following SQL statements:
  - ALTER
  - CALL
  - COMMENT
  - COMMIT
  - Compound SQL (compiled)
  - Compound SQL (inlined)
  - CREATE
  - DECLARE GLOBAL TEMPORARY TABLE
  - DELETE
  - DROP
  - EXPLAIN
  - FLUSH EVENT MONITOR
  - FLUSH PACKAGE CACHE
  - GRANT
  - INSERT
  - LOCK TABLE
  - MERGE
  - REFRESH TABLE
  - RELEASE SAVEPOINT
  - RENAME
  - REVOKE
  - ROLLBACK

- SAVEPOINT
- select-statement
- SET COMPILATION ENVIRONMENT
- SET CURRENT DECFLOAT ROUNDING MODE
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT FEDERATED ASYNCHRONY
- SET CURRENT IMPLICIT XMLPARSE OPTION
- SET CURRENT ISOLATION
- SET CURRENT LOCALE LC_MESSAGES
- SET CURRENT LOCALE LC_TIME
- SET CURRENT LOCK TIMEOUT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT MDC ROLLOUT MODE
- SET CURRENT OPTIMIZATION PROFILE
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET CURRENT TEMPORAL BUSINESS_TIME
- SET CURRENT TEMPORAL SYSTEM_TIME
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE (only if DYNAMICRULES run behavior is in effect for the package)
- SET INTEGRITY
- SET PASSTHRU
- SET PATH
- SET ROLE (only if DYNAMICRULES run behavior is in effect for the package)
- SET SCHEMA
- SET SERVER OPTION
- SET SESSION AUTHORIZATION
- SET SQL_CCFLAGS
- SET USAGE LIST STATE (only if DYNAMICRULES run behavior is in effect for the package)
- SET variable
- TRANSFER OWNERSHIP (only if DYNAMICRULES run behavior is in effect for the package)
- TRUNCATE (only if DYNAMICRULES run behavior is in effect for the package)
- UPDATE

Notes

- ***Parameter markers:*** Although a statement string cannot include references to host variables, it can include *parameter markers*. These can be replaced by the values of host variables when the prepared statement is executed. In the case of a CALL statement, a parameter marker can also be used for OUT and INOUT

arguments to the procedure. After the CALL is executed, the returned value for the argument will be assigned to the host variable corresponding to the parameter marker.

A parameter marker is a question mark (?) or a colon followed by a name (:name) that is used where a host variable could be used if the statement string were a static SQL statement. For an explanation of how parameter markers are replaced by values, see "OPEN" and "EXECUTE".

If the parameter marker is named, the name can include letters, numbers, and the symbols @, #, $, and _. The name is not folded to upper case.

Named parameter markers have the same syntax as host variables, but the two are not interchangeable. A host variable has a value and is used directly in a static SQL statement. A named parameter marker is a placeholder for a value in a dynamic SQL statement and the value is provided when the statement is executed.

There are two types of parameter markers:

Typed parameter marker
A parameter marker that is specified along with its target data type. It has the general form:

```
CAST(? AS data-type)
```

This notation is not a function call, but a "promise" that the type of the parameter at run time will be of the data type specified or some data type that can be converted to the specified data type. For example, in:

```
  UPDATE EMPLOYEE
SET LASTNAME = TRANSLATE(CAST(? AS VARCHAR(12)))
WHERE EMPNO = ?
```

the value of the argument of the TRANSLATE function will be provided at run time. The data type of that value will either be VARCHAR(12), or some type that can be converted to VARCHAR(12).

Untyped parameter marker
A parameter marker that is specified without its target data type. It has the form of a single question mark. The data type of an untyped parameter marker is provided by context. For example, the untyped parameter marker in the predicate of the previous update statement is the same as the data type of the EMPNO column.

Typed parameter markers can be used in dynamic SQL statements wherever a host variable is supported and the data type is based on the promise made in the CAST function.

Untyped parameter markers can be used in dynamic SQL statements as long as the data type of the parameter marker can be derived based on the context in the SQL statement (SQLSTATE 42610).

The following example results in an error since in the first context, *c1* would resolve to a string data type, but in the second context, *c1* would resolve to a numeric data type:

```
SELECT 'Hello' || c1, 5 + c1 FROM (VALUES(?)) AS T(c1)
```

However, the following statement is successful since the parameter marker associated with the derived column, *c1*, would resolve to a numeric data type for both contexts:

```
SELECT 7 + c1, 5 + c1 FROM (VALUES(?)) AS T(c1)
```

See "Determining data types of untyped expressions" for the rules for typing an untyped parameter marker.

- When a PREPARE statement is executed, the statement string is parsed and checked for errors. If the statement string is invalid, the error condition is reported in the SQLCA. Any subsequent EXECUTE or OPEN statement that references this statement will also receive the same error (due to an implicit prepare done by the system) unless the error has been corrected.
- Prepared statements can be referred to in the following kinds of statements, with the restrictions shown:

  In... The prepared statement...

  DESCRIBE
  can be any statement

  DECLARE CURSOR
  must be SELECT

  EXECUTE
  must *not* be SELECT
- A prepared statement can be executed many times. Indeed, if a prepared statement is not executed more than once and does not contain parameter markers, it is more efficient to use the EXECUTE IMMEDIATE statement rather than the PREPARE and EXECUTE statements.
- All prepared statements created by a unit of work remain in a prepared state until the application terminates, with the following exceptions:
  - A statement that is prepared within a package bound with `KEEPDYNAMC NO` and which is not used by an open cursor declared with the WITH HOLD option is no longer in a prepared state when the unit of work ends.
  - A dynamic statement that is bound with `KEEPDYNAMIC NO` and which is used by an open cursor declared with the WITH HOLD option is in a prepared state until the next unit of work boundary where the cursor is closed.

Examples

Example 1: Prepare and execute a non-select-statement in a COBOL program. Assume the statement is contained in a host variable HOLDER and that the program will place a statement string into the host variable based on some instructions from the user. The statement to be prepared does not have any parameter markers.

```
EXEC SQL  PREPARE STMT_NAME FROM :HOLDER
END-EXEC.
EXEC SQL  EXECUTE STMT_NAME
END-EXEC.
```

Example 2: Prepare and execute a non-select-statement as in example 1, except code it for a C program. Also assume the statement to be prepared can contain any number of parameter markers.

```
EXEC SQL  PREPARE STMT_NAME FROM :holder;
EXEC SQL  EXECUTE STMT_NAME USING DESCRIPTOR :insert_da;
```

Assume that the following statement is to be prepared:

```
INSERT INTO DEPT VALUES(?, ?, ?, ?)
```

The columns in the DEPT table are defined as follows:

```
DEPT_NO   CHAR(3) NOT NULL, -- department number
DEPTNAME  VARCHAR(29), -- department name
MGRNO     CHAR(6), -- manager number
ADMRDEPT  CHAR(3)  -- admin department number
```

To insert department number G01 named COMPLAINTS, which has no manager and reports to department A00, the structure INSERT_DA should have the values in Table 34 before issuing the EXECUTE statement.

Table 34. Required values for the INSERT_DA structure

SQLDA field	**Value**
SQLDAID	SQLDA
SQLDABC	192 (See note 1.)
SQLN	4
SQLD	4
SQLTYPE	452
SQLLEN	3
SQLDATA	*pointer to* G01
SQLIND	(See note 2.)
SQLNAME	
SQLTYPE	449
SQLLEN	29
SQLDATA	*pointer to* COMPLAINTS
SQLIND	*pointer to* 0
SQLNAME	
SQLTYPE	453
SQLLEN	6
SQLDATA	(See note 3.)
SQLIND	*pointer to* -1
SQLNAME	
SQLTYPE	453
SQLLEN	3
SQLDATA	*pointer to* A00
SQLIND	*pointer to* 0
SQLNAME	

Note:

1. This value is for a PREPARE done from a 32-bit application. If the PREPARE was done in a 64-bit application, then SQLDABC would have the value 240.
2. The value in SQLIND for this SQLVAR is ignored because the SQLTYPE identifies a non-nullable data type.
3. The value in SQLDATA for this SQLVAR is ignored because the value of SQLIND indicates this is a null value.

REFRESH TABLE

The REFRESH TABLE statement refreshes the data in a materialized query table.

Invocation

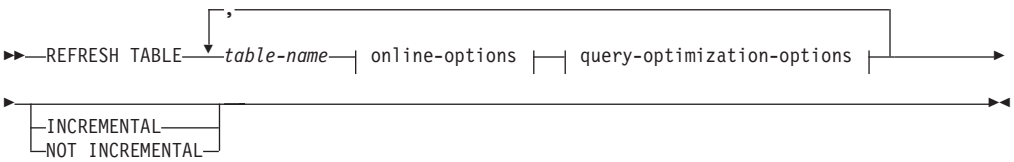
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the table
- DATAACCESS authority

Syntax



online-options:

query-optimization-options:

Description

table-name
Identifies the table to be refreshed.

The name, including the implicit or explicit schema, must identify a table that already exists at the current server. The table must allow the REFRESH TABLE statement (SQLSTATE 42809). This includes materialized query tables defined with:

- REFRESH IMMEDIATE
- REFRESH DEFERRED

online-options
Specifies the accessibility of the table while it is being processed.

ALLOW NO ACCESS
Specifies that no other users can access the table while it is being refreshed, except if they are using the Uncommitted Read isolation level.

ALLOW READ ACCESS
Specifies that other users have read-only access to the table while it is being refreshed.

ALLOW WRITE ACCESS
Specifies that other users have read and write access to the table while it is being refreshed.

To prevent a rollback of the entire statement because of a lock timeout when using the ALLOW READ ACCESS or the ALLOW WRITE ACCESS option, it is recommended that you issue a SET CURRENT LOCK TIMEOUT statement (specifying the WAIT option) before executing the REFRESH TABLE statement, and to reset the special register to its previous value afterwards. Note, however, that the CURRENT LOCK TIMEOUT register only impacts a specific set of lock types, not all lock types.

query-optimization-options
Specifies the query optimization options for the refresh of REFRESH DEFERRED materialized query tables.

ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY
Specifies that when the CURRENT REFRESH AGE special register is set to 'ANY', the refresh of *table-name* will allow REFRESH DEFERRED materialized query tables to be used to optimize the query that is used to refresh *table-name*. If *table-name* is not a REFRESH DEFERRED materialized query table, an error is returned (SQLSTATE 428FH). REFRESH IMMEDIATE materialized query tables are always considered for query optimization.

INCREMENTAL
Specifies an incremental refresh for the table by considering only the delta portion (if any) of its underlying tables or the content of an associated staging table (if one exists and its contents are consistent). If such a request cannot be satisfied (that is, the system detects that the materialized query table definition needs to be fully recomputed), an error (SQLSTATE 55019) is returned.

NOT INCREMENTAL
Specifies a full refresh for the table by recomputing the materialized query table definition.

If neither INCREMENTAL nor NOT INCREMENTAL is specified, the system will determine whether incremental processing is possible; if not, full refresh will be performed. If a staging table is present for the materialized query table that is to be refreshed, and incremental processing is not possible because the staging table is in a pending state, an error is returned (SQLSTATE 428A8). Full refresh will be performed if the staging table or the materialized query table is in an inconsistent state; otherwise, the contents of the staging table will be used for incremental processing.

Rules

- If REFRESH TABLE is issued on a materialized query table that references one or more nicknames, the authorization ID of the statement must have authority to select from the tables at the data source (SQLSTATE 42501).

Notes

- When the statement is used to refresh a REFRESH IMMEDIATE materialized query table whose underlying tables have been loaded, attached, or detached, the system might choose to incrementally refresh the materialized query table with the delta portions of its underlying tables. When the statement is used to refresh a REFRESH DEFERRED materialized query table with a supporting staging table, the system might choose to incrementally refresh the materialized query table with the delta portions of its underlying tables that have been captured in the staging table. However, there are some situations in which this optimization is not possible, and a full refresh (that is, a recomputation of the materialized query table definition) is necessary to ensure data integrity. You can explicitly request incremental maintenance by specifying the INCREMENTAL option; if this optimization is not possible, the system returns an error (SQLSTATE 55019).
- If the ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY option is used, ensure that the refresh order is correct for REFRESH DEFERRED materialized query tables. For example, consider two materialized query tables, MQT1 and MQT2, whose materialized queries share the same underlying tables. The materialized query for MQT2 can be calculated using MQT1, instead of the underlying tables. If separate statements are used to refresh these two materialized query tables, and MQT2 is refreshed first, the system might choose to use the contents of MQT1, which have not yet been refreshed, to refresh MQT2. In this case, MQT1 would contain current data, but MQT2 could still contain stale data, even though both were refreshed at almost the same time. The correct refresh order, if two REFRESH statements are used instead of one, is to refresh MQT1 first.
- If the materialized query table has an associated staging table, the staging table is pruned when the refresh is successfully performed.
- Any label-based access control on the base tables or on the materialized query table does not interfere with the refresh process. The refresh happens as if label-based access control were not present. The automatic protection that is associated with the materialized query table when it is created ensures that the data from the base tables remains protected when it is passed into the materialized query table.
- For materialized query table only, SET INTEGRITY FOR *mqt_name* IMMEDIATE CHECKED is the same as REFRESH TABLE *mqt_name*.
- **Refresh use of materialized query tables:** Materialized query tables are not used to evaluate the *select-statement* during the processing of the REFRESH TABLE statement.
- **Refresh isolation level:** The isolation level used to evaluate the *select-statement* is the isolation level specified on the *isolation-level* clause of the *select-statement*. Or, if the *isolation-level* clause was not specified, the isolation level of the materialized query table recorded when CREATE TABLE or ALTER TABLE was issued is used to evaluate the *select-statement*.
- Consider the statement:

  ```
  SET INTEGRITY FOR T IMMEDIATE CHECKED
  ```

  In the following scenarios, neither the INCREMENTAL check option for T nor an incremental refresh of T---if T is a materialized query table (MQT) or a staging table---is supported:

  – New constraints have been added to T while it is in set integrity pending state

  - When a LOAD REPLACE operation against T, it parents, or its underlying tables has taken place
  - When the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated after the last integrity check on T, its parents, or its underlying tables
  - The cascading effect of full processing, when any parent of T (or underlying table, if T is a materialized query table or a staging table) has been checked for integrity non-incrementally
  - If the table space containing the table or its parent (or underlying table of a materialized query table or a staging table) has been rolled forward to a point in time, and the table and its parent (or underlying table if the table is a materialized query table or a staging table) reside in different table spaces
  - T is an MQT, and a LOAD REPLACE or LOAD INSERT operation directly into T has taken place after the last refresh
- Incremental processing will be used whenever the situation allows it, because it is more efficient. The INCREMENTAL option is not needed in most cases. It is needed, however, to ensure that integrity checks are indeed processed incrementally. If the system detects that full processing is needed to ensure data integrity, an error is returned (SQLSTATE 55019).
- If the conditions for full processing described in the previous bullet are not satisfied, the system will perform an incremental refresh (if it is a materialized query table) when the user does not specify the NOT INCREMENTAL option for the statement `SET INTEGRITY FOR T IMMEDIATE CHECKED`.

RELEASE (connection)

The RELEASE (Connection) statement places one or more connections in the release-pending state.

Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax

Notes:

1 Note that an application server named CURRENT or ALL can only be identified by a host variable or a delimited identifier.

Description

server-name **or** *host-variable*
Identifies the application server by the specified *server-name* or a *host-variable* which contains the *server-name*.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-aligned and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

The specified database-alias or the database-alias contained in the host variable must identify an existing connection of the application process. If the database-alias does not identify an existing connection, an error (SQLSTATE 08003) is raised.

CURRENT
Identifies the current connection of the application process. The application process must be in the connected state. If not, an error (SQLSTATE 08003) is raised.

ALL or ALL SQL
Identifies all existing connections of the application process. This form of the RELEASE statement places all existing connections of the application process in the release-pending state. All connections will therefore be destroyed during

the next commit operation. An error or warning does not occur if no connections exist when the statement is executed.

Examples

- *Example 1:* The SQL connection to IBMSTHDB is no longer needed by the application. The following statement will cause it to be destroyed during the next commit operation:

```
EXEC SQL RELEASE IBMSTHDB;
```

- *Example 2:* The current connection is no longer needed by the application. The following statement will cause it to be destroyed during the next commit operation:

```
EXEC SQL RELEASE CURRENT;
```

- *Example 3:* If an application has no need to access the databases after a commit but will continue to run for a while, then it is better not to tie up those connections unnecessarily. The following statement can be executed before the commit to ensure all connections will be destroyed at the commit:

```
EXEC SQL RELEASE ALL;
```

RELEASE SAVEPOINT

The RELEASE SAVEPOINT statement is used to indicate that the application no longer wishes to have the named savepoint maintained. After this statement has been invoked, rollback to the savepoint is no longer possible.

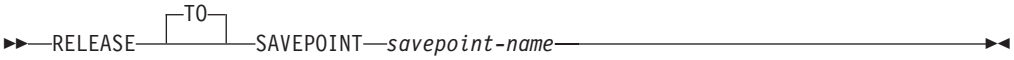
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

savepoint-name
Specifies the savepoint that is to be released. Any savepoints nested within the named savepoint are also released. Rollback to that savepoint, or any savepoint nested within it, is no longer possible. If the named savepoint does not exist in the current savepoint level (see the "Rules" section in the description of the SAVEPOINT statement), an error is returned (SQLSTATE 3B001). The specified *savepoint-name* cannot begin with 'SYS' (SQLSTATE 42939).

Notes

- The name of the savepoint that was released can now be reused in another SAVEPOINT statement, regardless of whether the UNIQUE keyword was specified on an earlier SAVEPOINT statement specifying this same savepoint name.

Example

Release a savepoint named SAVEPOINT1.

```
RELEASE SAVEPOINT SAVEPOINT1
```

RENAME

The RENAME statement renames an existing table or index.

Invocation

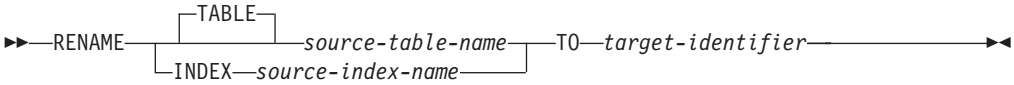
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the table or index
- Ownership of the table or index, as recorded in the OWNER column of the SYSCAT.TABLES catalog view for a table, and the SYSCAT.INDEXES catalog view for an index
- ALTERIN privilege on the schema
- DBADM authority

Syntax



Description

TABLE *source-table-name*

Names the existing table that is to be renamed. The name, including the schema name, must identify a table that already exists in the database (SQLSTATE 42704). It must not be the name of a catalog table (SQLSTATE 42832), a materialized query table, a typed table (SQLSTATE 42997), a created temporary table, a declared global temporary table (SQLSTATE 42995), a nickname, or an object other than a table or an alias (SQLSTATE 42809). The TABLE keyword is optional.

The name must not identify a table that is referenced in a row permission definition or a column mask definition (SQLSTATE 42917).

INDEX *source-index-name*

Names the existing index that is to be renamed. The name, including the schema name, must identify an index that already exists in the database (SQLSTATE 42704). It must not be the name of an index on a created temporary table or a declared global temporary table (SQLSTATE 42995). The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42832).

target-identifier

Specifies the new name for the table or index without a schema name. The schema name of the source object is used to qualify the new name for the object. The qualified name must *not* identify a table, view, alias, or index that already exists in the database (SQLSTATE 42710).

Rules

When renaming a table, the source table must not:

- Be referenced in any existing materialized query table definitions
- Be the subject table of an existing trigger
- Be a parent or dependent table in any referential integrity constraints
- Be the scope of any existing reference column
- Be referenced by an XSR object that has been enabled for decomposition

An error (SQLSTATE 42986) is returned if the source table violates one or more of these conditions.

When renaming an index:

- The source index must not be a system-generated index for an implementation table on which a typed table is based (SQLSTATE 42858).

Notes

- Catalog entries are updated to reflect the new table or index name.
- *All* authorizations associated with the source table or index name are *transferred* to the new table or index name (the authorization catalog tables are updated appropriately).
- Indexes defined over the source table are *transferred* to the new table (the index catalog tables are updated appropriately).
- RENAME TABLE invalidates any packages that are dependent on the source table. RENAME INDEX invalidates any packages that are dependent on the source index.
- If an alias is used for the *source-table-name*, it must resolve to a table name. The table is renamed within the schema of this table. The alias is not changed by the RENAME statement and continues to refer to the old table name.
- A table with primary key or unique constraints can be renamed if none of the primary key or unique constraints are referenced by any foreign key.

Examples

- *Example 1*: Change the name of the EMP table to EMPLOYEE.

```
RENAME TABLE EMP TO EMPLOYEE
RENAME TABLE ABC.EMP TO EMPLOYEE
```

- *Example 2*: Change the name of the index NEW-IND to IND.

```
RENAME INDEX NEW-IND TO IND
RENAME INDEX ABC.NEW-IND TO IND
```

RENAME STOGROUP

The RENAME STOGROUP statement renames an existing storage group.

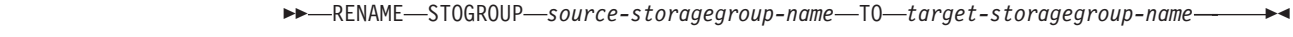
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include either SYSCTRL or SYSADM authority.

Syntax



Description

source-storagegroup-name
Identifies the storage group to rename; *source-storagegroup-name* must identify a storage group that exists at the current server (SQLSTATE 42704). This is a one-part name.

target-storagegroup-name
Names the storage group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *target-storagegroup-name* must not identify a storage group that already exists in the catalog (SQLSTATE 42710). The *target-storagegroup-name* must not begin with the characters 'SYS' (SQLSTATE 42939).

Rules

- The RENAME STOGROUP statement cannot be executed while a database partition server is being added (SQLSTATE 55071).

RENAME TABLESPACE

The RENAME TABLESPACE statement renames an existing table space.

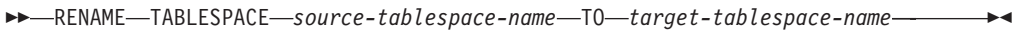
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include either SYSCTRL or SYSADM authority.

Syntax



Description

source-tablespace-name
Specifies the existing table space that is to be renamed, as a one-part name. It is an SQL identifier (either ordinary or delimited). The table space name must identify a table space that already exists in the catalog (SQLSTATE 42704).

target-tablespace-name
Specifies the new name for the table space, as a one-part name. It is an SQL identifier (either ordinary or delimited). The new table space name must *not* identify a table space that already exists in the catalog (SQLSTATE 42710), and it cannot start with 'SYS' (SQLSTATE 42939).

Rules

- The SYSCATSPACE table space cannot be renamed (SQLSTATE 42832).
- Any table spaces with "rollforward pending" or "rollforward in progress" states cannot be renamed (SQLSTATE 55039)

Notes

- Renaming a table space will update the minimum recovery time of a table space to the point in time when the rename took place. This implies that a roll forward at the table space level must be to at least this point in time.
- The new table space name must be used when restoring a table space from a backup image, where the rename was done after the backup was created.

Example

Change the name of the table space USERSPACE1 to DATA2000:

```
RENAME TABLESPACE USERSPACE1 TO DATA2000
```

REPEAT

The REPEAT statement executes a statement or group of statements until a search condition is true.

Invocation

This statement can be embedded in an:

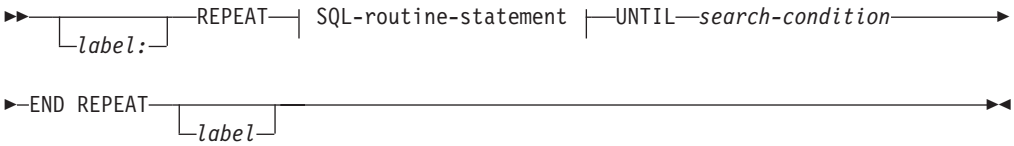
- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

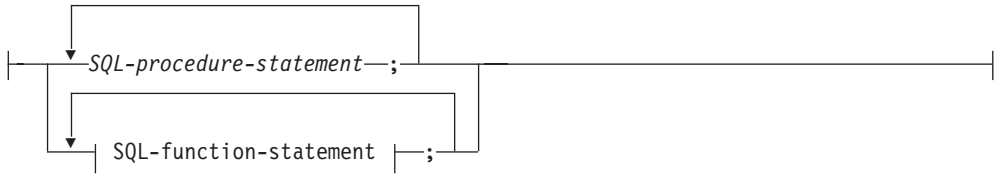
Authorization

No privileges are required to invoke the REPEAT statement. However, the authorization ID of the statement must hold the necessary privileges to invoke the SQL statements and search condition that are embedded in the REPEAT statement.

Syntax



SQL-routine-statement:



Description

label
Specifies the label for the REPEAT statement. If the beginning label is specified, that label can be specified on LEAVE and ITERATE statements. If an ending label is specified, a matching beginning label also must be specified.

SQL-procedure-statement
Specifies the SQL statements to execute within the loop. *SQL-procedure-statement* is only applicable when in the context of an SQL procedure or a compound SQL (compiled) statement. See *SQL-procedure-statement* in "Compound SQL (compiled)" statement.

SQL-function-statement
Specifies the SQL statements to execute within the loop. *SQL-function-statement* is only applicable when in the context of an SQL trigger, SQL function, or SQL method. See *SQL-function-statement* in "FOR".

search-condition
The *search-condition* is evaluated after each execution of the REPEAT loop. If the condition is true, the loop will exit. If the condition is unknown or false, the looping continues.

Example

A REPEAT statement fetches rows from a table until the *not_found* condition handler is invoked.

```
CREATE PROCEDURE REPEAT_STMT(OUT counter INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE v_firstnme VARCHAR(12);
    DECLARE v_midinit CHAR(1);
    DECLARE v_lastname VARCHAR(15);
    DECLARE at_end SMALLINT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
      SELECT firstnme, midinit, lastname
        FROM employee;
    DECLARE CONTINUE HANDLER FOR not_found
      SET at_end = 1;
    OPEN c1;
    fetch_loop:
    REPEAT
      FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
      SET v_counter = v_counter + 1;
      UNTIL at_end > 0
    END REPEAT fetch_loop;
    SET counter = v_counter;
    CLOSE c1;
  END
```

RESIGNAL

The RESIGNAL statement is used within a condition handler to resignal the condition that activated the handler, or to raise an alternate condition so that it can be processed at a higher level. It causes an exception, warning, or not found condition to be returned, along with optional message text.

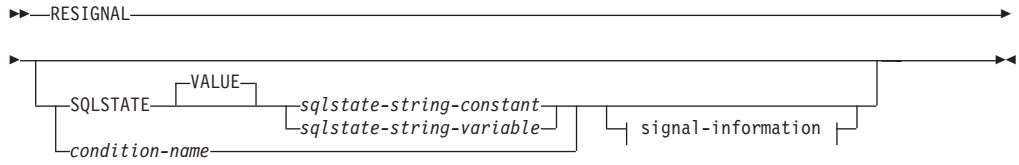
Invocation

This statement can only be embedded in a condition handler within a compound SQL (compiled) statement. The compound SQL (compiled) statement can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition.

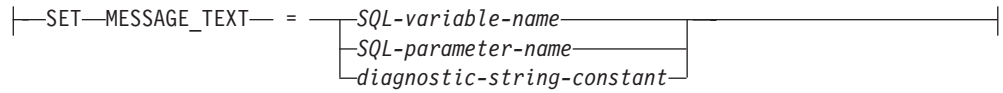
Authorization

If a module condition is referenced, the privileges held by the authorization ID of the statement must include EXECUTE privilege on the module.

Syntax



signal-information:



Description

SQLSTATE VALUE *sqlstate-string-constant*

The specified string constant represents an SQLSTATE. It must be a character string constant with exactly 5 characters that follow the rules for SQLSTATEs:

- Each character must be from the set of digits ('0' through '9') or non-accented upper case letters ('A' through 'Z')
- The SQLSTATE class (first two characters) cannot be '00', since this represents successful completion.

If the SQLSTATE does not conform to these rules, an error is raised (SQLSTATE 428B3).

SQLSTATE VALUE

Specifies the SQLSTATE that will be returned. Any valid SQLSTATE value can be used. The specified value must follow the rules for SQLSTATEs:

- Each character must be from the set of digits ('0' through '9') or upper case letters ('A' through 'Z') without diacritical marks
- The SQLSTATE class (first two characters) cannot be '00', since this represents successful completion.

If the SQLSTATE does not conform to these rules, an error is returned.

sqlstate-string-constant
The *sqlstate-string-constant* must be a character string constant with exactly 5 characters.

sqlstate-string-variable
The specified SQL variable or SQL parameter must be of data type CHAR(5) and must not be the null value.

condition-name
Specifies the name of a condition that will be returned. The *condition-name* must be declared within the compound-statement or identify a condition that exists at the current server.

SET MESSAGE_TEXT =
Specifies a string that describes the error or warning. The string is returned in the `sqlerrmc` field of the SQLCA. If the actual string is longer than 70 bytes, it is truncated without warning.

SQL-variable-name
Identifies an SQL variable, declared within the compound statement, that contains the message text.

SQL-parameter-name
Identifies an SQL parameter, defined for the routine, that contains the message text. The SQL parameter must be defined as a CHAR or VARCHAR data type.

diagnostic-string-constant
Specifies a character string constant that contains the message text.

Notes

- If a RESIGNAL statement is issued without specifying an SQLSTATE clause or a *condition-name*, the identical condition that invoked the handler is returned. The SQLSTATE, SQLCODE and the SQLCA associated with the condition are unchanged.
- If a RESIGNAL statement is issued using a *condition-name* that has no associated SQLSTATE value and the condition is not handled, SQLSTATE 45000 is returned and the SQLCODE is set to -438. Note that such a condition will not be handled by a condition handler for SQLSTATE 45000 that is within the scope of the routine issuing the RESIGNAL statement.
- If a RESIGNAL statement is issued using an SQLSTATE value or a *condition-name* with an associated SQLSTATE value, the SQLCODE returned is based on the SQLSTATE value as follows:
  - If the specified SQLSTATE class is either '01' or '02', a warning or not found condition is returned and the SQLCODE is set to +438.
  - Otherwise, an exception condition is returned and the SQLCODE is set to -438.
- A RESIGNAL statement has the indicated fields of the SQLCA set as follows:
  - `sqlerrd` fields are set to zero
  - `sqlwarn` fields are set to blank
  - `sqlerrmc` is set to the first 70 bytes of MESSAGE_TEXT
  - `sqlerrml` is set to the length of `sqlerrmc`, or to zero if no SET MESSAGE_TEXT clause is specified
  - `sqlerrp` is set to ROUTINE
- Refer to the "Notes" section under "SIGNAL statement" for further information about SQLSTATE values.

Example

This example detects a division by zero error. The IF statement uses a SIGNAL statement to invoke the *overflow* condition handler. The condition handler uses a RESIGNAL statement to return a different SQLSTATE value to the client application.

```
CREATE PROCEDURE divide ( IN numerator INTEGER,
                          IN denominator INTEGER,
                          OUT result INTEGER)
LANGUAGE SQL
BEGIN
  DECLARE overflow CONDITION FOR SQLSTATE '22003';
  DECLARE CONTINUE HANDLER FOR overflow
    RESIGNAL SQLSTATE '22375';
  IF denominator = 0 THEN
    SIGNAL overflow;
  ELSE
    SET result = numerator / denominator;
  END IF;
END
```

RETURN

The RETURN statement is used to return from a routine. For SQL functions or methods, it returns the result of the function or method. For an SQL procedure, it optionally returns an integer status value.

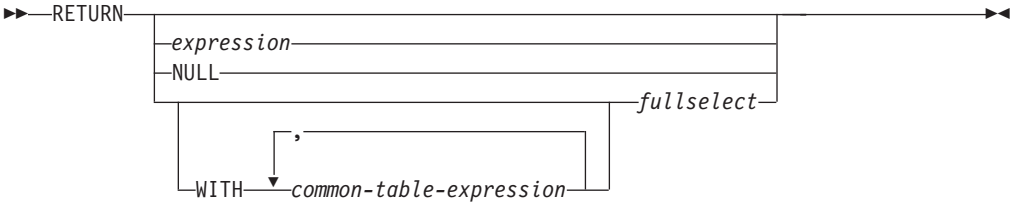
Invocation

This statement can be embedded in an SQL function, SQL method, or SQL procedure. It is not an executable statement and cannot be dynamically prepared.

Authorization

No privileges are required to invoke the RETURN statement. However, the authorization ID of the statement must hold the necessary privileges to invoke any expression or fullselect that is embedded in the RETURN statement.

Syntax



Description

expression
Specifies a value that is returned from the routine:

- If the routine is a function or method other than a compiled table function, one of *expression*, NULL, or *fullselect* must be specified (SQLSTATE 42631) and the data type of the result must be assignable to the RETURNS type of the routine (SQLSTATE 42866).
- If the routine is an inlined table function, a scalar expression (other than a scalar fullselect) cannot be specified (SQLSTATE 428F1). If the routine is a compiled table function, an expression cannot be specified.
- If the routine is a procedure, the data type of *expression* must be INTEGER (SQLSTATE 428F2). A procedure cannot return NULL or a *fullselect*.

NULL
Specifies that the function or method returns a null value of the data type defined in the RETURNS clause. NULL cannot be specified for a RETURN from a table function, row function, or procedure.

WITH *common-table-expression*
Defines a common table expression for use with the *fullselect* that follows.

fullselect
Specifies the row or rows to be returned for the function. The number of columns in the *fullselect* must match the number of columns in the function result (SQLSTATE 42811). In addition, the static column types of the *fullselect* must be assignable to the declared column types of the function result, using the rules for assignment to columns (SQLSTATE 42866).

The *fullselect* cannot be specified for a RETURN from a procedure or a compiled table function.

If the routine is a scalar function or method, then the *fullselect* must return one column (SQLSTATE 42823) and, at most, one row (SQLSTATE 21000).

If the routine is a row function, it must return, at most, one row (SQLSTATE 21505). However, one or more columns can be returned.

If the routine is an inlined table function, it can return zero or more rows with one or more columns. If the fullselect has zero result rows, no row is returned to the result table by the RETURN statement.

Rules

- The execution of an SQL function or method must end with a RETURN statement (SQLSTATE 42632).
- In an SQL table function using a compound SQL (compiled) statement, an *expression*, NULL, or *fullselect*cannot be specified. Rows are returned from the function using the PIPE statement and the RETURN statement is required as the last statement to execute when the function exits (SQLSTATE 2F005).
- In an SQL table or row function using a compound SQL (inlined) statement, the only RETURN statement allowed is the one at the end of the compound statement. (SQLSTATE 429BD).

Notes

- When a value is returned from a procedure, the caller can access the value:
  - using the GET DIAGNOSTICS statement to retrieve the DB2_RETURN_STATUS when the SQL procedure was called from another SQL procedure
  - using the parameter bound for the return value parameter marker in the escape clause CALL syntax (?=CALL...) in a CLI application
  - directly from the `sqlerrd[0]` field of the SQLCA, after processing the CALL of an SQL procedure. This field is only valid if the SQLCODE is zero or positive (assume a value of -1 otherwise).

Example

Use a RETURN statement to return from an SQL procedure with a status value of zero if successful, and -200 if not.

```
BEGIN
...
  GOTO FAIL;
...
  SUCCESS: RETURN 0;
  FAIL: RETURN -200;
END
```

REVOKE (database authorities)

This form of the REVOKE statement revokes authorities that apply to the entire database.

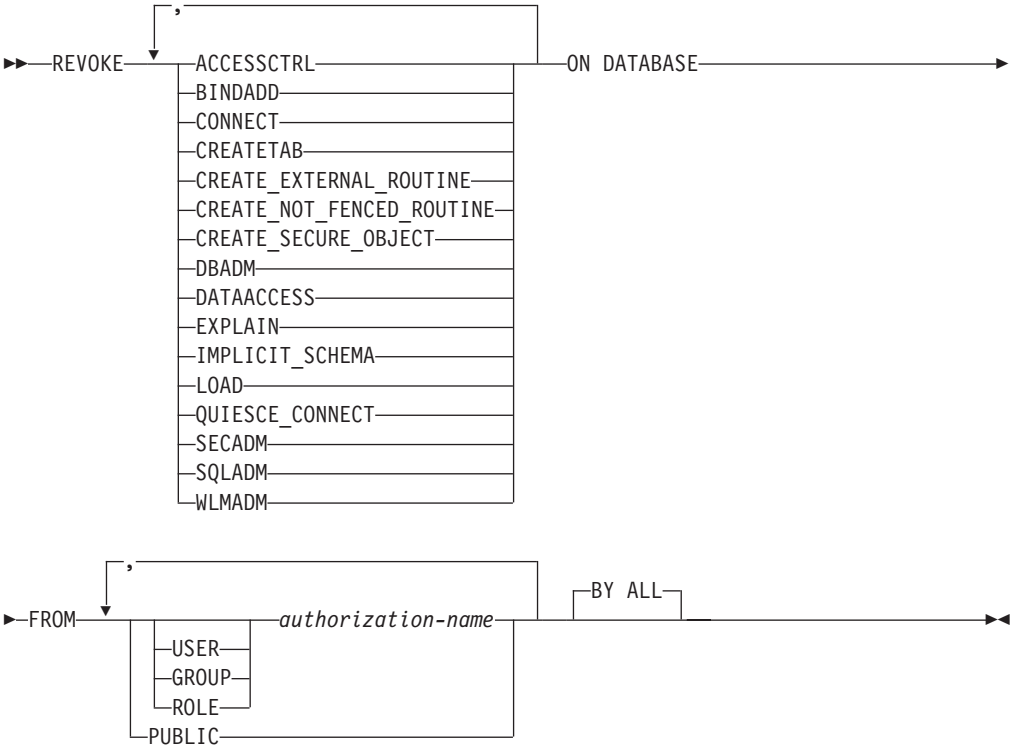
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

To revoke ACCESSCTRL, CREATE_SECURE_OBJECT, DATAACCESS, DBADM, or SECADM authority, SECADM authority is required. To revoke other authorities, ACCESSCTRL or SECADM authority is required.

Syntax



Description

ACCESSCTRL
Revokes the authority to grant and revoke most database authorities and object privileges.

BINDADD
Revokes the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if his BINDADD authority is subsequently revoked.

The BINDADD authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority.

CONNECT
Revokes the authority to access the database.

Revoking the CONNECT authority from a user does not affect any privileges that were granted to that user on objects in the database. If the user is subsequently granted the CONNECT authority again, all previously held privileges are still valid (assuming they were not explicitly revoked).

The CONNECT authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

CREATETAB
Revokes the authority to create tables. The creator of a table automatically has the CONTROL privilege on that table, and retains this privilege even if his CREATETAB authority is subsequently revoked.

The CREATETAB authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

CREATE_EXTERNAL_ROUTINE
Revokes the authority to register external routines. Once an external routine has been registered, it continues to exist, even if CREATE_EXTERNAL_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

CREATE_EXTERNAL_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM or CREATE_NOT_FENCED_ROUTINE authority without also revoking DBADM or CREATE_NOT_FENCED_ROUTINE authority (SQLSTATE 42504).

CREATE_NOT_FENCED_ROUTINE
Revokes the authority to register routines that execute in the database manager's process. Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE_NOT_FENCED_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

CREATE_NOT_FENCED_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

CREATE_SECURE_OBJECT
Revokes the authority to create secure triggers and secure functions. Revokes the authority to alter the secure attribute of such objects as well.

DATAACCESS
Revokes the authority to access data.

DBADM
Revokes the DBADM authority.

DBADM authority cannot be revoked from PUBLIC (because it cannot be granted to PUBLIC).

CAUTION:
Revoking DBADM authority does not automatically revoke any privileges that were held by the *authorization-name* on objects in the database.

EXPLAIN
Revokes the authority to explain, prepare, and describe static and dynamic statements without requiring access to data.

IMPLICIT_SCHEMA
Revokes the authority to implicitly create a schema. It does not affect the ability to create objects in existing schemas or to process a CREATE SCHEMA statement.

IMPLICIT_SCHEMA authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

LOAD
Revokes the authority to LOAD in this database.

QUIESCE_CONNECT
Revokes the authority to access the database while it is quiesced.

SECADM
Revokes the authority to administer database security.

SQLADM
Revokes the authority to monitor and tune SQL statements.

WLMADM
Revokes the authority to manage workload manager objects.

FROM
Indicates from whom the authorities are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the authorities from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules

Security administrator mandatory: The database must have at least one authorization ID of type USER with the SECADM authority. The SECADM authority cannot be revoked from every user authorization ID (SQLSTATE 42523).

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.DBAUTH catalog view where the grantee is *authorization-name*:

- If all rows have a GRANTEETYPE of 'U', USER is assumed.
- If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
- If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
- If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- Revoking a specific privilege does not necessarily revoke the ability to perform an action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds a higher level authority, such as DBADM.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products.
  - CREATE_NOT_FENCED can be specified in place of CREATE_NOT_FENCED_ROUTINE
  - SYSTEM can be specified in place of DATABASE
  - NOT INCLUDING DEPENDENT PRIVILEGES may be specified as a syntax alternative

Examples

- *Example 1:* Given that USER6 is only a user and not a group, revoke the privilege to create tables from the user USER6.

```
REVOKE CREATETAB ON DATABASE FROM USER6
```

- *Example 2:* Revoke BINDADD authority on the database from a group named D024. There are two rows in the SYSCAT.DBAUTH catalog view for this grantee; one with a GRANTEETYPE of U and one with a GRANTEETYPE of G.

```
REVOKE BINDADD ON DATABASE FROM GROUP D024
```

  In this case, the GROUP keyword must be specified; otherwise an error will occur (SQLSTATE 56092).

- *Example 3:* Revoke security administrator authority from user Walid.

```
REVOKE SECADM ON DATABASE FROM USER Walid
```

- *Example 4:* A user with SECADM authority revokes the CREATE_SECURE_OBJECT authority from user Haytham.

```
REVOKE CREATE_SECURE_OBJECT ON DATABASE FROM USER HAYTHAM
```

REVOKE (exemption)

This form of the REVOKE statement revokes an exemption to a label-based access control (LBAC) access rule.

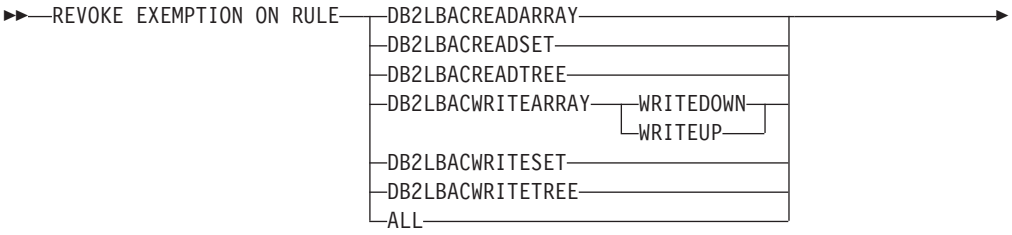
Invocation

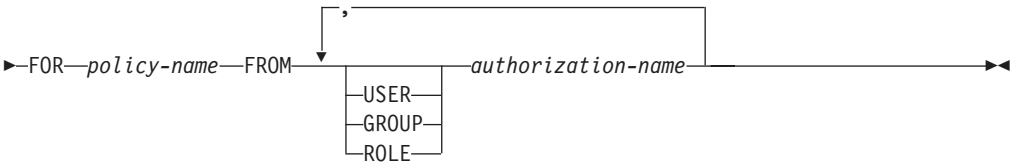
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax





Description

EXEMPTION ON RULE
Revokes the exemption on an access rule.

DB2LBACREADARRAY
Revokes an exemption on the predefined DB2LBACREADARRAY rule.

DB2LBACREADSET
Revokes an exemption on the predefined DB2LBACREADSET rule.

DB2LBACREADTREE
Revokes an exemption on the predefined DB2LBACREADTREE rule.

DB2LBACWRITEARRAY
Revokes an exemption on the predefined DB2LBACWRITEARRAY rule.

WRITEDOWN
Specifies that the exemption only applies to write down.

WRITEUP
Specifies that the exemption only applies to write up.

DB2LBACWRITESET
Revokes an exemption on the predefined DB2LBACWRITESET rule.

DB2LBACWRITETREE
Revokes an exemption on the predefined DB2LBACWRITETREE rule.

ALL
Revokes the exemptions on all of the predefined rules.

FOR *policy-name*
Specifies the name of the security policy on which exemptions are to be revoked.

FROM
Specifies from whom the exemption is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.SECURITYPOLICYEXEMPTIONS catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Examples

- *Example 1:* Revoke the exemption on access rule DB2LBACREADSET for security policy DATA_ACCESS from user WALID.

```
REVOKE EXEMPTION ON RULE DB2LBACREADSET FOR DATA_ACCESS
  FROM USER WALID
```

- *Example 2:* Revoke an exemption on access rule DB2LBACWRITEARRAY with the WRITEDOWN option for security policy DATA_ACCESS from user BOBBY.

```
REVOKE EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEDOWN
  FOR DATA_ACCESS FROM USER BOBBY
```

- *Example 3:* Revoke an exemption on access rule DB2LBACWRITEARRAY with the WRITEUP option for security policy DATA_ACCESS from user BOBBY.

```
REVOKE EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEUP
  FOR DATA_ACCESS FROM USER BOBBY
```

REVOKE (global variable privileges)

This form of the REVOKE statement revokes one or more privileges on a created global variable.

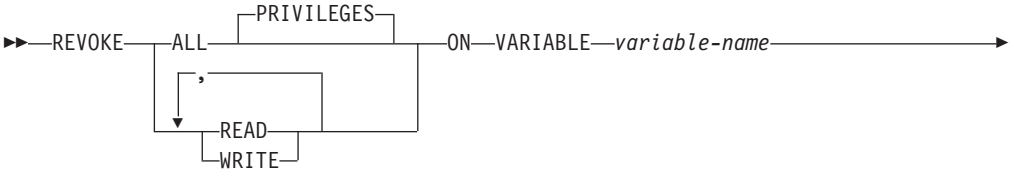
Invocation

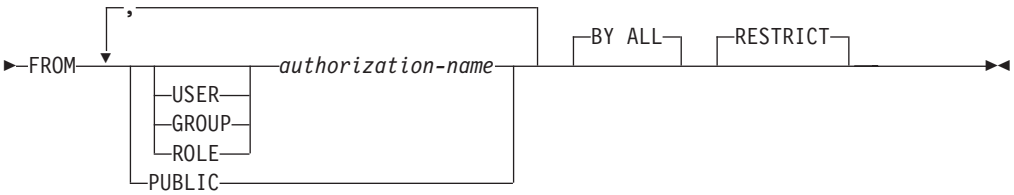
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax





Description

ALL PRIVILEGES
Revokes all privileges held by an *authorization-name* for the specified global variable. If ALL is not specified, READ or WRITE must be specified. READ or WRITE must not be specified more than once.

READ
Revokes the privilege to read the value of the specified global variable.

WRITE
Revokes the privilege to assign a value to the specified global variable.

ON VARIABLE *variable-name*
Identifies the global variable on which one or more privileges are to be revoked. The *variable-name* must identify a global variable that exists at the current server and is not a module variable (SQLSTATE 42704).

FROM
Specifies from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

ROLE
Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the specified privileges from PUBLIC.

BY ALL
Revokes each specified privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

RESTRICT
Specifies that the statement is to fail if any objects depend on the privileges being revoked. This is the default behavior.

Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified, then for all rows for the specified object in the SYSCAT.VARIABLEAUTH catalog view where the grantee is *authorization-name*:
  - If GRANTEETYPE is 'U', USER is assumed.
  - If GRANTEETYPE is 'G', GROUP is assumed.
  - If GRANTEETYPE is 'R', ROLE is assumed.
  - If GRANTEETYPE does not have the same value, an error is returned (SQLSTATE 56092.
- If any SQL function, SQL method, procedure, view, trigger, or another global variable contains a global variable and depends on the privilege being revoked, the revoke operation will fail (SQLSTATE 42893).

Notes

- If the READ privilege on a global variable is revoked, packages with a dependency to write the value of the global variable (for example, by the SET statement) are not affected, because writing to a global variable is controlled by the WRITE privilege on that global variable.
- If the WRITE privilege on a global variable is revoked, packages with a dependency to read the value of the global variable are not affected, because reading from a global variable is controlled by the READ privilege on that global variable.
- Revoking a privilege does not necessarily impair the ability to perform the action. A user might be able to proceed if the required privilege is held through membership in a different group or role, or by PUBLIC.

Example

Revoke the WRITE privilege on global variable MYSCHEMA.MYJOB_PRINTER from user ZUBIRI.

```
REVOKE WRITE ON VARIABLE MYSCHEMA.MYJOB_PRINTER FROM ZUBIRI
```

REVOKE (index privileges)

This form of the REVOKE statement revokes the CONTROL privilege on an index.

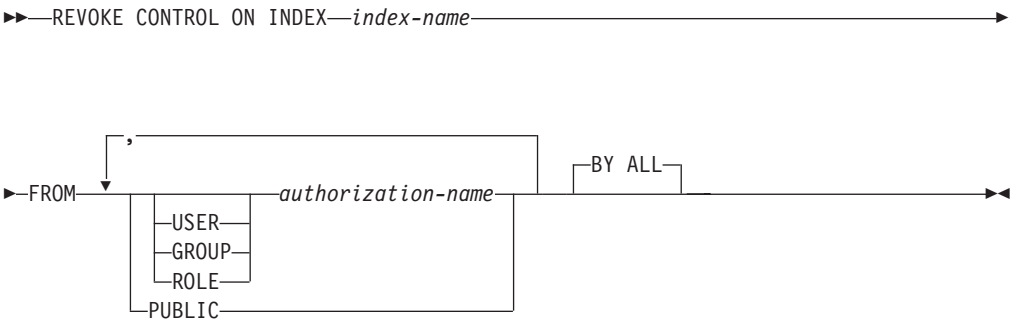
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax



Description

CONTROL
Revokes the privilege to drop the index. This is the CONTROL privilege for indexes, which is automatically granted to creators of indexes.

ON INDEX *index-name*
Specifies the name of the index on which the CONTROL privilege is to be revoked.

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.INDEXAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds authorities such as ALTERIN on the schema of an index.

Examples

- *Example 1:* Given that USER4 is only a user and not a group, revoke the privilege to drop an index DEPTIDX from the user USER4.

```
REVOKE CONTROL ON INDEX DEPTIDX FROM KIESLER
```

- *Example 2:* Revoke the privilege to drop an index LUNCHITEMS from the user CHEF and the group WAITERS.

```
REVOKE CONTROL ON INDEX LUNCHITEMS
  FROM USER CHEF, GROUP WAITERS
```

REVOKE (module privileges)

This form of the REVOKE statement revokes the privilege on a module.

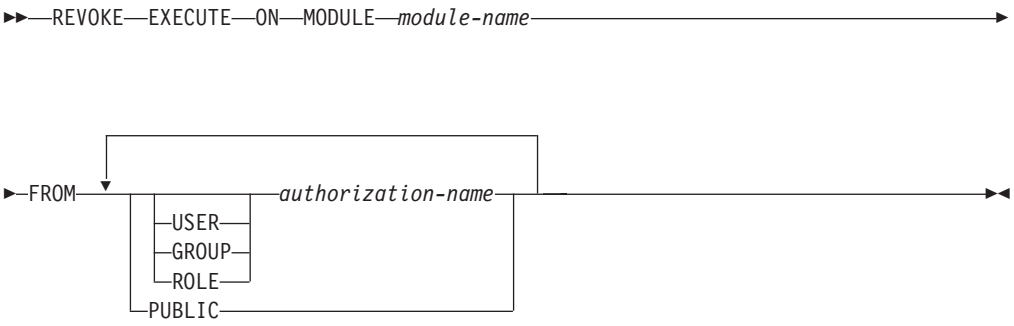
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax



Description

EXECUTE
Revokes the privilege to reference published module objects. This includes revoking the privilege to:

- Execute any published routine defined in the module.
- Read from and write to any published global variables defined in the module.
- Reference any published user-defined types defined in the module.
- Reference any published conditions defined in the module.

ON MODULE *module-name*
Identifies the module on which the privilege is revoked. The *module-name* must identify a module that exists at the current server (SQLSTATE 42704).

FROM
Indicates from whom the privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name
Lists one or more authorization IDs. The same *authorization-name* must not be specified more than once

PUBLIC
Grants the privilege to a set of users (authorization IDs). For more information, see "Authorization, privileges and object ownership".

Example

The following example demonstrate how to revoke the EXECUTE privilege from a module named *myModa* from user *jones*

```
REVOKE EXECUTE ON MODULE MYMODA FROM JONES
```

REVOKE (package privileges)

This form of the REVOKE statement revokes CONTROL, BIND, and EXECUTE privileges against a package.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

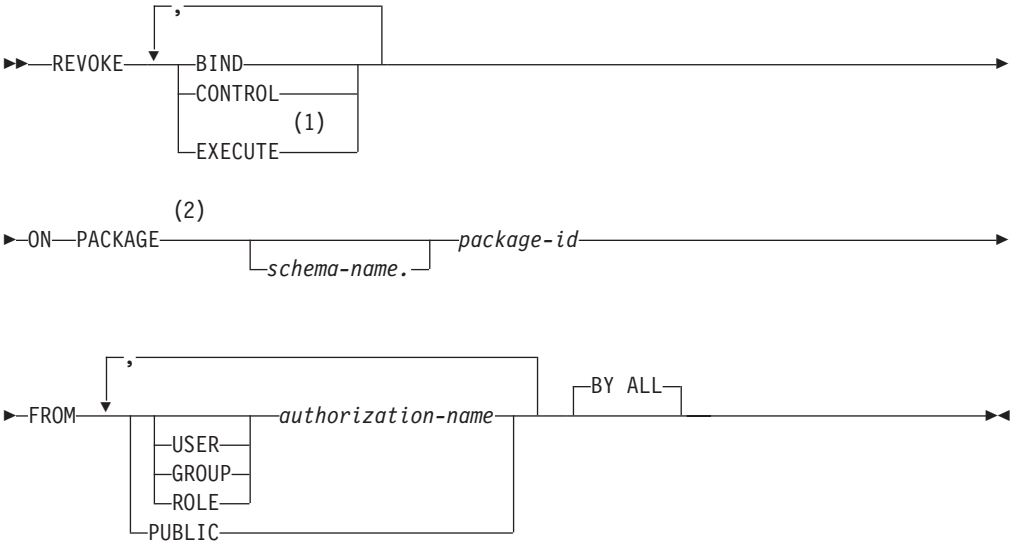
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the referenced package
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to revoke the CONTROL privilege.

Syntax



Notes:

1 RUN can be used as a synonym for EXECUTE.

2 PROGRAM can be used as a synonym for PACKAGE.

Description

BIND

Revokes the privilege to execute BIND or REBIND on-or to add a new version of- the referenced package.

The BIND privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package, without also revoking the CONTROL privilege.

CONTROL
Revokes the privilege to drop the package and to extend package privileges to other users.

Revoking CONTROL does not revoke the other package privileges.

EXECUTE
Revokes the privilege to execute the package.

The EXECUTE privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package without also revoking the CONTROL privilege.

ON PACKAGE *schema-name.package-id*
Specifies the name of the package on which privileges are to be revoked. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The revoking of a package privilege applies to all versions of the package.

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.PACKAGEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds privileges such as ALTERIN on the schema of a package.

Examples

- *Example 1:* Revoke the EXECUTE privilege on package CORPDATA.PKGA from PUBLIC.

```
REVOKE EXECUTE
  ON PACKAGE CORPDATA.PKGA
  FROM PUBLIC
```

- *Example 2:* Revoke CONTROL authority on the RRSP_PKG package for the user FRANK and for PUBLIC.

```
REVOKE CONTROL
  ON PACKAGE RRSP_PKG
  FROM USER FRANK, PUBLIC
```

REVOKE (role)

This form of the REVOKE statement revokes roles from users, groups, or other roles.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

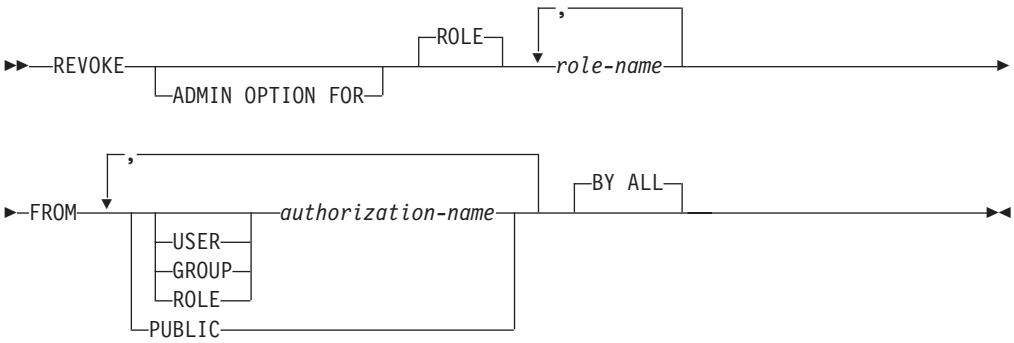
Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- The WITH ADMIN OPTION on the role
- SECADM authority

SECADM authority is required to revoke the ADMIN OPTION FOR *role-name* from an *authorization-name* or to revoke a *role-name* from an *authorization-name* that has the WITH ADMIN OPTION on that role.

Syntax



Description

ADMIN OPTION FOR
Revokes the WITH ADMIN OPTION on *role-name*. The WITH ADMIN OPTION on *role-name* must be held by *authorization-name* or by PUBLIC, if PUBLIC is specified (SQLSTATE 42504). If the ADMIN OPTION FOR clause is specified, only the WITH ADMIN OPTION on ROLE *role-name* is revoked, not the role itself.

ROLE *role-name*
Specifies the role that is to be revoked. The *role-name* must identify an existing role at the current server (SQLSTATE 42704) that has been granted to *authorization-name* or to PUBLIC, if PUBLIC is specified (SQLSTATE 42504).

FROM
Specifies from whom the role is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

ROLE
Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the specified roles from PUBLIC.

BY ALL
Revokes the *role-name* from each specified *authorization-name* that was explicitly granted that role, regardless of who granted it. This is the default behavior.

Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified, then for all rows for the specified object in the SYSCAT.ROLEAUTH catalog view where the grantee is *authorization-name*:
  - If GRANTEETYPE is 'U', USER is assumed.
  - If GRANTEETYPE is 'G', GROUP is assumed.
  - If GRANTEETYPE is 'R', ROLE is assumed.
  - If GRANTEETYPE does not have the same value, an error is returned (SQLSTATE 56092.
- The *role-name* must not identify a role, or a role that contains *role-name*, if the role has either EXECUTE privilege on a routine or USAGE privilege on a sequence, and an SQL object other than a package is dependent on the routine or sequence (SQLSTATE 42893). The owner of the SQL object is either *authorization-name* or any user that is a member of *authorization-name*, where *authorization-name* is a role.

Notes

- If a role is revoked from an *authorization-name* or from PUBLIC, all privileges that the role held are no longer available to the *authorization-name* or to PUBLIC through that role.
- Revoking a role does not necessarily revoke the ability to perform a particular action by way of a privilege that was granted to that role. A user might still be able to proceed if other privileges are held by PUBLIC, by a group to which the user belongs, by another role granted to the user, or if the user has a higher level authority, such as DBADM.

Examples

- *Example 1:* Revoke the role INTERN from the role DOCTOR and the role DOCTOR from the role SPECIALIST.

```
REVOKE ROLE INTERN FROM ROLE DOCTOR

REVOKE ROLE DOCTOR FROM ROLE SPECIALIST
```

- *Example 2:* Revoke the role INTERN from PUBLIC.

```
REVOKE ROLE INTERN FROM PUBLIC
```

- *Example 3:* Revoke the role SPECIALIST from user BOB and group TORONTO.

```
REVOKE ROLE SPECIALIST FROM USER BOB, GROUP TORONTO BY ALL
```

REVOKE (routine privileges)

This form of the REVOKE statement revokes privileges on a routine (function, method, or procedure) that is not defined in a module.

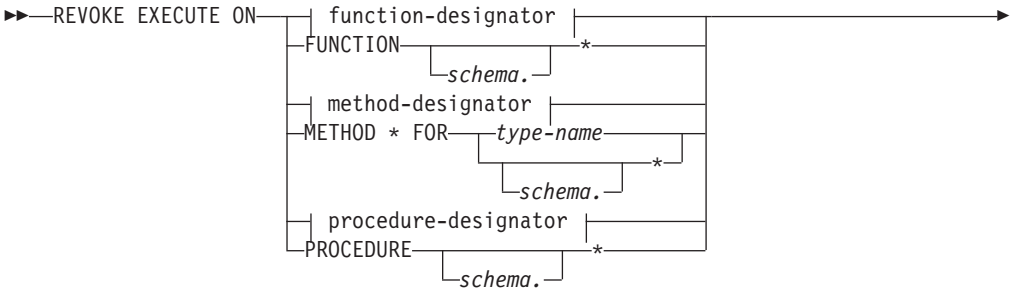
Invocation

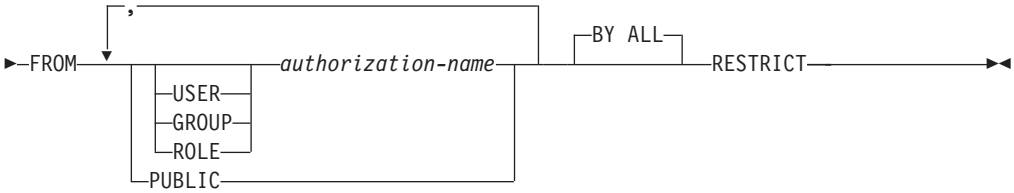
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

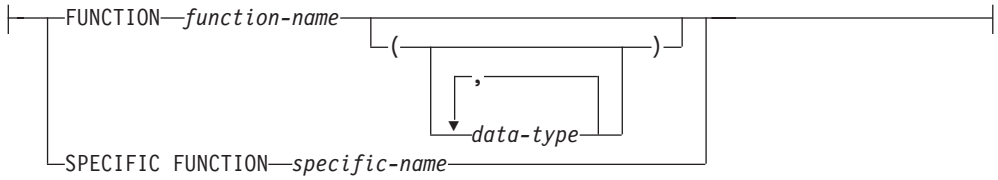
The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax

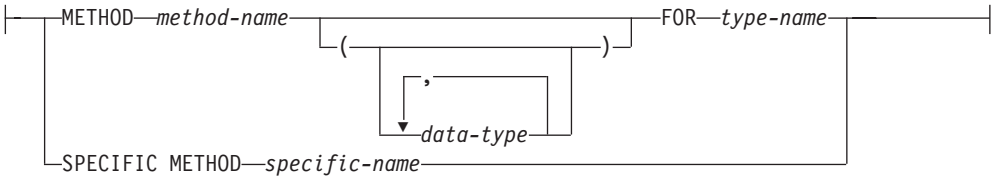




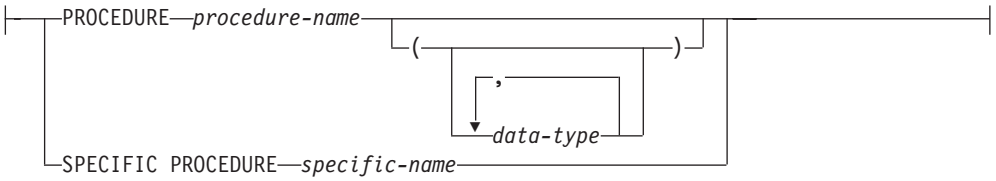
function-designator:



method-designator:



procedure-designator:



Description

EXECUTE
Revokes the privilege to run the identified user-defined function, method, or procedure.

function-designator
Uniquely identifies the function from which the privilege is revoked. For more information, see "Function, method, and procedure designators" on page 20.

FUNCTION *schema.**
Identifies the explicit grant for all the existing and future functions in the schema. Revoking the *schema*.* privilege does not revoke any privileges that were granted on a specific function. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

method-designator
Uniquely identifies the method from which the privilege is revoked. For more information, see "Function, method, and procedure designators" on page 20.

METHOD *
Identifies the explicit grant for all the existing and future methods for the type *type-name*. Revoking the * privilege does not revoke any privileges that were granted on a specific method.

FOR *type-name*
Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names. An asterisk (*) can be used in place of *type-name* to identify the explicit grant on all existing and future methods for all existing and future types in the schema. Revoking the privilege using an asterisk for method and *type-name* does not revoke any privileges that were granted on a specific method or on all methods for a specific type.

procedure-designator
Uniquely identifies the procedure from which the privilege is revoked. For more information, see "Function, method, and procedure designators" on page 20.

PROCEDURE *schema.**
Identifies the explicit grant for all the existing and future procedures in the schema. Revoking the *schema*.* privilege does not revoke any privileges that were granted on a specific procedure. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

FROM
Specifies from whom the EXECUTE privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the EXECUTE privilege from PUBLIC.

BY ALL
Revokes the EXECUTE privilege from all named users who were explicitly granted the privilege, regardless of who granted it. This is the default behavior.

RESTRICT
Specifies that the EXECUTE privilege cannot be revoked if both of the following conditions are true (SQLSTATE 42893):

- The specified routine is used in a view, trigger, constraint, index extension, SQL function, SQL method, transform group, or is referenced as the SOURCE of a sourced function.
- The loss of the EXECUTE privilege would cause the owner of the view, trigger, constraint, index extension, SQL function, SQL method, transform group, or sourced function to no longer be able to execute the specified routine.

Rules

- It is not possible to revoke the EXECUTE privilege on a function or method defined with schema 'SYSIBM' or 'SYSFUN' (SQLSTATE 42832).
- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.ROUTINEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.

- If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- If a package depends on a routine (function, method, or procedure), and the EXECUTE privilege on that routine is revoked from PUBLIC, a user, or a role, the package becomes inoperative if the routine is a function or a method, and the package becomes invalid if the routine is a procedure, unless the package owner still holds the EXECUTE privilege on the routine. The package owner can still hold the EXECUTE privilege if:
  - The package owner was explicitly granted the EXECUTE privilege
  - The package owner is a member of a role that holds the EXECUTE privilege
  - The EXECUTE privilege was granted to PUBLIC

  Because group privileges are not considered for static packages, the package becomes inoperative (in the case of a function or a method) or invalid (in the case of a procedure) even if a group to which the package owner belongs holds the EXECUTE privilege.

Examples

- *Example 1:* Revoke the EXECUTE privilege on function CALC_SALARY from user JONES. Assume that there is only one function in the schema with function name CALC_SALARY.

```
REVOKE EXECUTE ON FUNCTION CALC_SALARY FROM JONES RESTRICT
```

- *Example 2:* Revoke the EXECUTE privilege on procedure VACATION_ACCR from all users at the current server.

```
REVOKE EXECUTE ON PROCEDURE VACATION_ACCR FROM PUBLIC RESTRICT
```

- *Example 3:* Revoke the EXECUTE privilege on function NEW_DEPT_HIRES from HR (Human Resources). The function has two input parameters of type INTEGER and CHAR(10), respectively. Assume that the schema has more than one function named NEW_DEPT_HIRES.

```
REVOKE EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10))
  FROM HR RESTRICT
```

- *Example 4:* Revoke the EXECUTE privilege on method SET_SALARY for type EMPLOYEE from user Jones.

```
REVOKE EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE FROM JONES RESTRICT
```

REVOKE (schema privileges)

This form of the REVOKE statement revokes the privileges on a schema.

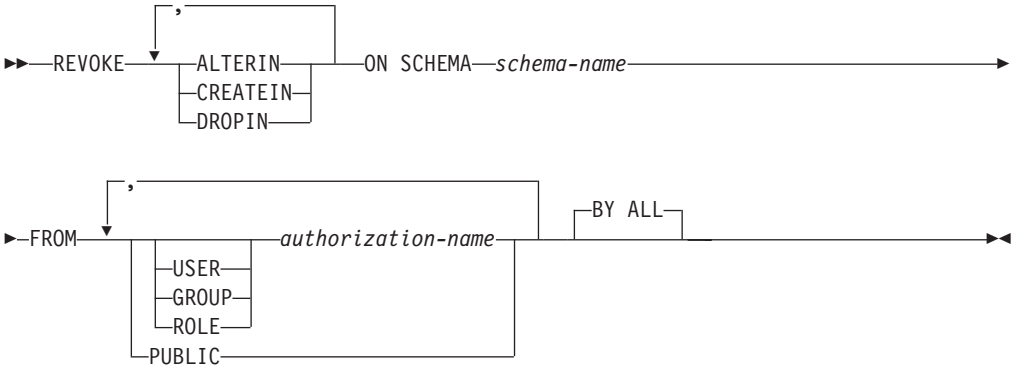
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax



Description

ALTERIN
Revokes the privilege to alter or comment on objects in the schema.

CREATEIN
Revokes the privilege to create objects in the schema.

DROPIN
Revokes the privilege to drop objects in the schema.

ON SCHEMA *schema-name*
Specifies the name of the schema on which privileges are to be revoked.

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.SCHEMAAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds a higher level authority such as DBADM.

Examples

- *Example 1:* Given that USER4 is only a user and not a group, revoke the privilege to create objects in schema DEPTIDX from the user USER4.

```
REVOKE CREATEIN ON SCHEMA DEPTIDX FROM USER4
```

- *Example 2:* Revoke the privilege to drop objects in schema LUNCH from the user CHEF and the group WAITERS.

```
REVOKE DROPIN ON SCHEMA LUNCH
  FROM USER CHEF, GROUP WAITERS
```

REVOKE (security label)

This form of the REVOKE statement revokes a label-based access control (LBAC) security label.

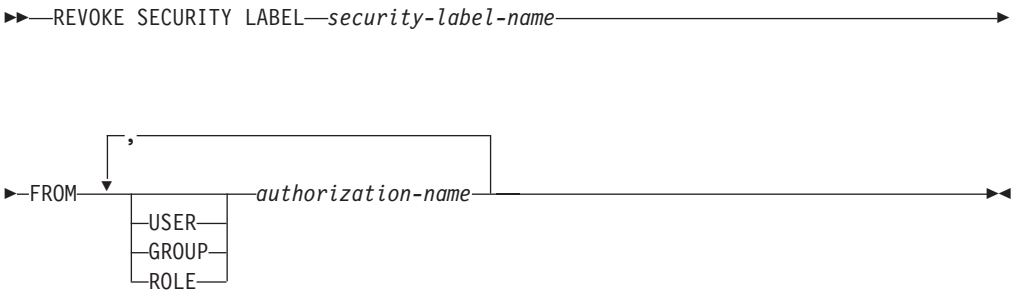
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

SECURITY LABEL *security-label-name*
Revokes the security label *security-label-name*. The name must be qualified with a security policy (SQLSTATE 42704) and must identify a security label that exists at the current server (SQLSTATE 42704), and that is held by *authorization-name* (SQLSTATE 42504).

FROM
Specifies from whom the specified security label is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name. The role name must exist at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:

- For all rows for the specified object in the SYSCAT.SECURITYLABELACCESS catalog view where the grantee is *authorization-name*:
  - If all rows have a GRANTEETYPE of 'U', USER is assumed.
  - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
  - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
  - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Example

Revoke the security label EMPLOYEESECLABEL, which is part of the security policy DATA_ACCESS, from user WALID.

```
REVOKE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
  FROM USER WALID
```

REVOKE (sequence privileges)

This form of the REVOKE statement revokes privileges on a sequence.

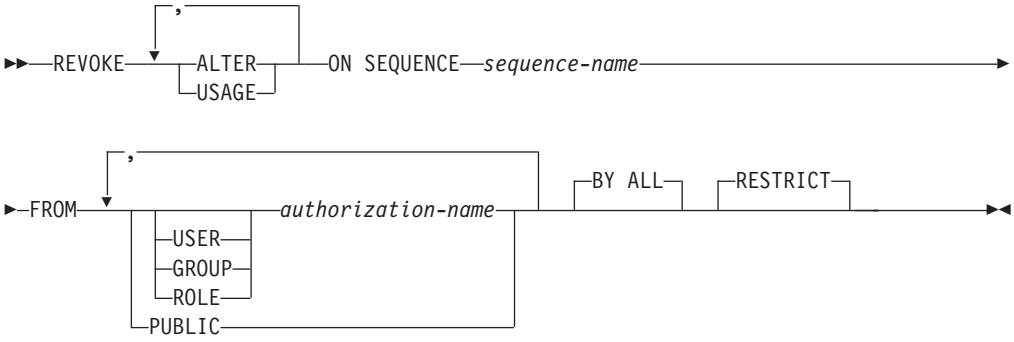
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax



Description

ALTER
Revokes the privilege to change the properties of a sequence or to restart sequence number generation using the ALTER SEQUENCE statement.

USAGE
Revokes the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

ON SEQUENCE *sequence-name*
Identifies the sequence on which the specified privileges are to be revoked. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error is returned (SQLSTATE 42704).

FROM
Specifies from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the specified privileges from PUBLIC.

BY ALL
Revokes each specified privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

RESTRICT
This optional keyword indicates that the statement will fail if any objects depend on the privilege being revoked.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.SEQUENCEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- Revoking a privilege on a sequence from the authorization ID under which a package was bound will cause the package to become invalid if the authorization ID does not continue to hold the privilege on the sequence through different means; for example, through membership in a role that holds the privilege.
- Revoking a specific privilege does not necessarily remove the ability to perform an action. A user can proceed if other privileges are held by PUBLIC or by a group to which the user belongs, or if the user has a higher level of authority, such as DBADM.

Examples

- *Example 1:* Revoke the USAGE privilege on a sequence called GENERATE_ID from user ENGLES. There is one row in the SYSCAT.SEQUENCEAUTH catalog view for this sequence and grantee, and the GRANTEETYPE value is U.

```
REVOKE USAGE ON SEQUENCE GENERATE_ID FROM ENGLES
```

- *Example 2:* Revoke alter privileges on sequence GENERATE_ID that were previously granted to all local users. (Grants to specific users are not affected.)

```
REVOKE ALTER ON SEQUENCE GENERATE_ID FROM PUBLIC
```

- *Example 3:* Revoke all privileges on sequence GENERATE_ID from users PELLOW and MLI, and from group PLANNERS.

```
REVOKE ALTER, USAGE ON SEQUENCE GENERATE_ID
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

REVOKE (server privileges)

This form of the REVOKE statement revokes the privilege to access and use a specified data source in pass-through mode.

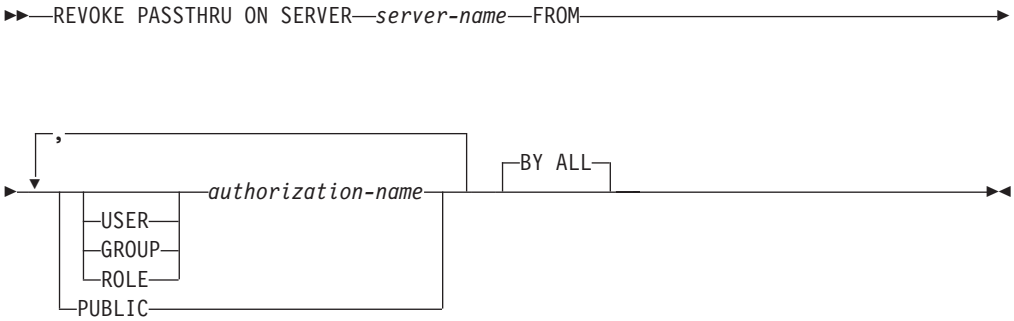
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL or SECADM authority.

Syntax



Description

SERVER *server-name*
Names the data source for which the privilege to use in pass-through mode is being revoked. *server-name* must identify a data source that is described in the catalog.

FROM
Specifies from whom the privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes from PUBLIC the privilege to pass through to *server-name*.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.PASSTHRUAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Examples

- *Example 1:* Revoke USER6's privilege to pass through to data source MOUNTAIN.

```
REVOKE PASSTHRU ON SERVER MOUNTAIN FROM USER USER6
```

- *Example 2:* Revoke group D024's privilege to pass through to data source EASTWING.

```
REVOKE PASSTHRU ON SERVER EASTWING FROM GROUP D024
```

The members of group D024 will no longer be able to use their group ID to pass through to EASTWING. But if any members have the privilege to pass through to EASTWING under their own user IDs, they will retain this privilege.

REVOKE (SETSESSIONUSER privilege)

This form of the REVOKE statement revokes one or more SETSESSIONUSER privileges from one or more authorization IDs.

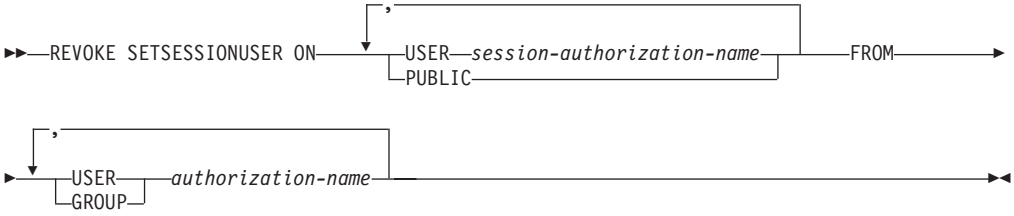
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax



Description

SETSESSIONUSER ON
Revokes the privilege to assume the identity of a new authorization ID.

USER *session-authorization-name*
Specifies the authorization ID that the *authorization-name* is able to assume, using the SET SESSION AUTHORIZATION statement. The *session-authorization-name* must identify a user that the *authorization-name* can assume, not a group (SQLSTATE 42504).

PUBLIC
Specifies that all privileges to set the session authorization will be revoked.

FROM
Specifies from whom the privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

Examples

- *Example 1:* User PAUL holds the privilege to set the session authorization to WALID and therefore to execute SQL statements as user WALID. The following statement revokes that privilege.

```
REVOKE SETSESSIONUSER ON USER WALID
  FROM USER PAUL
```

- *Example 2:* User GUYLAINE holds the privilege to set the session authorization to BOBBY, RICK, or KEVIN and therefore to execute SQL statements as BOBBY, RICK, or KEVIN. The following statement revokes the privilege to use two of those authorization IDs. After this statement executes, GUYLAINE will only be able to set the session authorization to KEVIN.

```
REVOKE SETSESSIONUSER ON USER BOBBY, USER RICK
  FROM USER GUYLAINE
```

- *Example 3:* The group ACCTG and user WALID can set session authorization to any authorization ID. The following statement revokes that privilege from both ACCTG and WALID.

```
REVOKE SETSESSIONUSER ON PUBLIC
  FROM USER WALID, GROUP ACCTG
```

REVOKE (table space privileges)

This form of the REVOKE statement revokes the USE privilege on a table space.

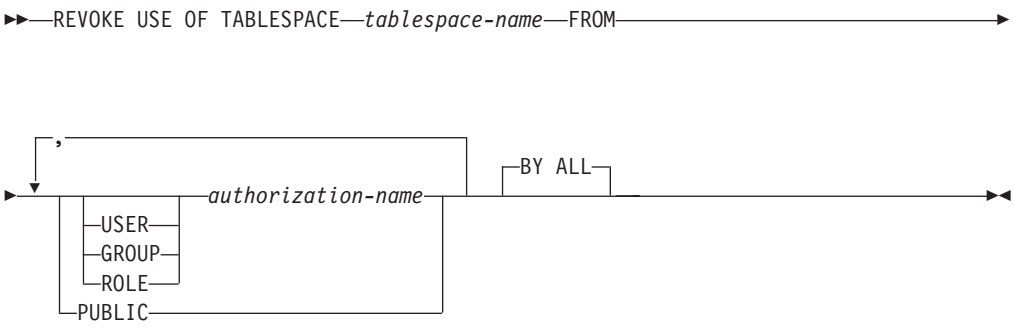
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL, SECADM, SYSCTRL, or SYSADM authority.

Syntax



Description

USE
Revokes the privilege to specify or default to the table space when creating a table.

OF TABLESPACE *tablespace-name*
Specifies the table space on which the USE privilege is to be revoked. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a SYSTEM TEMPORARY table space (SQLSTATE 42809).

FROM
Indicates from whom the USE privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the USE privilege from PUBLIC.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.TBSPACEAUTH catalog view where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- Revoking the USE privilege does not necessarily revoke the ability to create tables in that table space. A user may still be able to create tables in that table space if the USE privilege is held by PUBLIC or a group, or if the user has a higher level authority, such as DBADM.

Example

Revoke the privilege to create tables in table space PLANS from the user BOBBY.

```
REVOKE USE OF TABLESPACE PLANS FROM USER BOBBY
```

REVOKE (table, view, or nickname privileges)

This form of the REVOKE statement revokes privileges on a table, view, or nickname.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

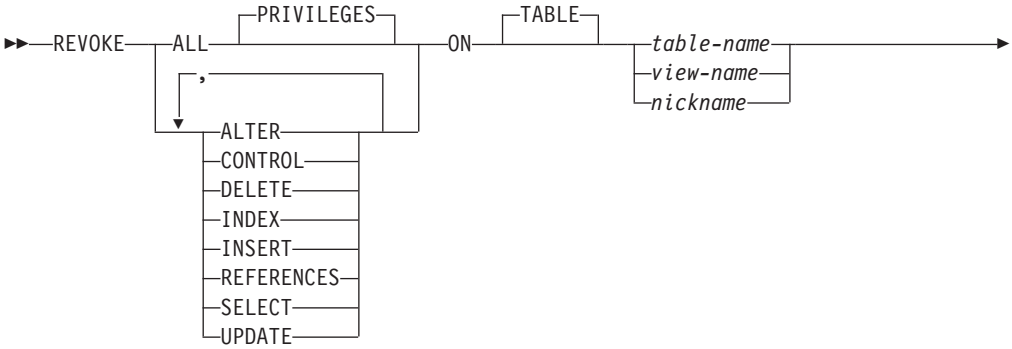
Authorization

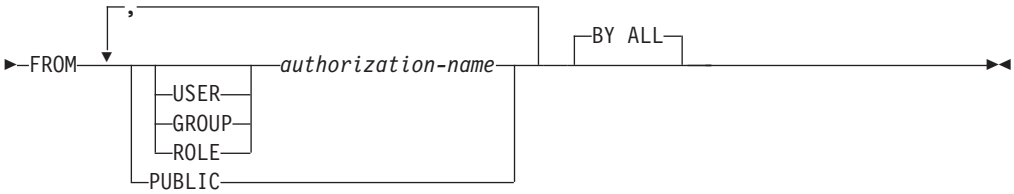
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the referenced table, view, or nickname
- ACCESSCTRL or SECADM authority

ACCESSCTRL or SECADM authority is required to revoke the CONTROL privilege, or to revoke privileges on catalog tables and views.

Syntax





Description

ALL or ALL PRIVILEGES

Revokes all privileges (except CONTROL) held by an authorization-name for the specified tables, views, or nicknames.

If ALL is not used, one or more of the keywords listed in the option stack (ALTER through UPDATE) must be used. Each keyword revokes the privilege described, but only as it applies to the tables, views, or nicknames named in the ON clause. The same keyword must not be specified more than once.

ALTER
Revokes the privilege to add columns to the base table definition; create or drop a primary key or unique constraint on the table; create or drop a foreign key on the table; add/change a comment on the table, view, or nickname; create or drop a check constraint; create a trigger; add, reset, or drop a column option for a nickname; or, change nickname column names or data types.

CONTROL
Revokes the ability to drop the table, view, or nickname, and the ability to execute the RUNSTATS utility on the table and indexes.

Revoking CONTROL privilege from an *authorization-name* does not revoke other privileges granted to the user on that object.

DELETE
Revokes the privilege to delete rows from the table, updatable view, or nickname.

INDEX
Revokes the privilege to create an index on the table or an index specification on the nickname. The creator of an index or index specification automatically has the CONTROL privilege over the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains this privilege even if the INDEX privilege is revoked.

INSERT
Revokes the privileges to insert rows into the table, updatable view, or nickname, and to run the IMPORT utility.

REFERENCES
Revokes the privilege to create or drop a foreign key referencing the table as the parent. Any column level REFERENCES privileges are also revoked.

SELECT
Revokes the privilege to retrieve rows from the table or view, to create a view on a table, and to run the EXPORT utility against the table or view.

Revoking SELECT privilege may cause some views to be marked inoperative. (For information about inoperative views, see "CREATE VIEW".)

UPDATE
Revokes the privilege to update rows in the table, updatable view, or nickname. Any column level UPDATE privileges are also revoked.

ON TABLE *table-name* **or** *view-name* **or** *nickname*
Specifies the table, view, or nickname on which privileges are to be revoked. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

ROLE
Specifies that the *authorization-name* identifies a role name.

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules

- For each *authorization-name* specified, if neither USER, GROUP, nor ROLE is specified, then:
  - For all rows for the specified object in the SYSCAT.TABAUTH and SYSCAT.COLAUTH catalog views where the grantee is *authorization-name*:
    - If all rows have a GRANTEETYPE of 'U', USER is assumed.
    - If all rows have a GRANTEETYPE of 'G', GROUP is assumed.
    - If all rows have a GRANTEETYPE of 'R', ROLE is assumed.
    - If all rows do not have the same value for GRANTEETYPE, an error is returned (SQLSTATE 56092).

Notes

- If a privilege is revoked from the *authorization-name* that is the owner of the view (as recorded in the OWNER column in SYSCAT.VIEWS), that privilege is also revoked from any dependent views.
- If the owner of the view loses a SELECT privilege on some object on which the view definition depends (or an object upon which the view definition depends is dropped, or made inoperative in the case of another view), the view will be made inoperative.

  However, if a user who holds ACCESSCTRL or SECADM authority explicitly revokes all privileges on the view from the owner, then the record of the OWNER will not appear in SYSCAT.TABAUTH but nothing will happen to the view - it remains operative.
- Privileges on inoperative views cannot be revoked.
- A package might become invalid when the authorization ID under which the package was bound loses a privilege on an object on which the package depends. The privilege can be lost in one of the following ways:
  - The privilege is revoked from the authorization ID
  - The privilege is revoked from a role of which the authorization ID is a member
  - The privilege is revoked from PUBLIC

  A package remains invalid until a bind or rebind operation on the application is successfully executed, or the application is executed and the database manager successfully rebinds the application (using information stored in the catalogs). Packages marked invalid due to a revoke may be successfully rebound without any additional grants.

  For example, if a package owned by USER1 contains a SELECT from table T1, and the SELECT privilege on table T1 is revoked from USER1, the package will be marked invalid. If SELECT authority is granted again, or if the user holds DBADM authority, the package is successfully rebound when executed.

Another example is a package owned by USER1, who is a member of role R1. The package contains a SELECT from table T1, and the SELECT privilege on table T1 is revoked from role R1. The package will be marked invalid, assuming USER1 does not hold the SELECT privilege on table T1 by other means.

- Packages, triggers or views that include the use of OUTER(*Z*) in the FROM clause, are dependent on having SELECT privilege on every subtable or subview of *Z*. Similarly, packages, triggers, or views that include the use of DEREF(*Y*) where *Y* is a reference type with a target table or view *Z*, are dependent on having SELECT privilege on every subtable or subview of *Z*. Such packages might become invalid, and such triggers or views made inoperative when the authorization ID under which the packages were bound, or the owner of the triggers or views loses the SELECT privilege. The SELECT privilege can be lost in one of the following ways:
  - SELECT privilege is revoked from the authorization ID
  - SELECT privilege is revoked from a role of which the authorization ID is a member
  - SELECT privilege is revoked from PUBLIC
- Table, view, or nickname privileges cannot be revoked from an *authorization-name* with CONTROL on the object without also revoking the CONTROL privilege (SQLSTATE 42504).
- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user can proceed with a task if other privileges are held by PUBLIC, a group, or a role, or if the user holds privileges such as ALTERIN on the schema of a table or a view.
- If the owner of the materialized query table loses a SELECT privilege on a table on which the materialized query table definition depends (or a table upon which the materialized query table definition depends is dropped), the materialized query table will be dropped.

  However, if a user who holds SECADM or ACCESSCTRL authority explicitly revokes all privileges on the materialized query table from the owner, then the record in SYSTABAUTH for the OWNER will be deleted, but nothing will happen to the materialized query table - it remains operative.
- Revoking nickname privileges has no affect on data source object (table or view) privileges.
- Revoking the SELECT privilege for a table or view that is directly or indirectly referenced in an SQL function or method body may fail if the SQL function or method body cannot be dropped because some other object is dependent on it (SQLSTATE 42893).
- Revoking the SELECT privilege causes an SQL function or method body to be dropped when:
  - The owner of the SQL function or method body loses the SELECT privilege on some object on which the SQL function or method body definition depends; note that the privilege can be lost because of a revoke from PUBLIC or from a role of which the owner is a member
  - An object on which the SQL function or method body definition depends is dropped

  However, the revoke fails if another object depends on the function or method (SQLSTATE 42893).
- **Revoking WITH GRANT OPTION:** The only way to revoke the WITH GRANT OPTION is to revoke the privilege itself and then grant it again without specifying WITH GRANT OPTION.

- **Revoking column privileges:** The only way to revoke column privileges is to revoke the privilege from the entire table itself and then grant it again for each column.

Examples

- *Example 1:* Revoke SELECT privilege on table EMPLOYEE from user ENGLES. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
  ON TABLE EMPLOYEE
  FROM ENGLES
```

- *Example 2:* Revoke update privileges on table EMPLOYEE previously granted to all local users. Note that grants to specific users are not affected.

```
REVOKE UPDATE
  ON EMPLOYEE
  FROM PUBLIC
```

- *Example 3:* Revoke all privileges on table EMPLOYEE from users PELLOW and MLI and from group PLANNERS.

```
REVOKE ALL
  ON EMPLOYEE
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

- *Example 4:* Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a user named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

  or

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM USER JOHN
```

  Note that an attempt to revoke the privilege from GROUP JOHN would result in an error, since the privilege was not previously granted to GROUP JOHN.

- *Example 5:* Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a group named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is G.

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

  or

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM GROUP JOHN
```

- *Example 6:* Revoke user SHAWN's privilege to create an index specification on nickname ORAREM1.

```
REVOKE INDEX
  ON ORAREM1 FROM USER SHAWN
```

REVOKE (workload privileges)

This form of the REVOKE statement revokes the USAGE privilege on a workload.

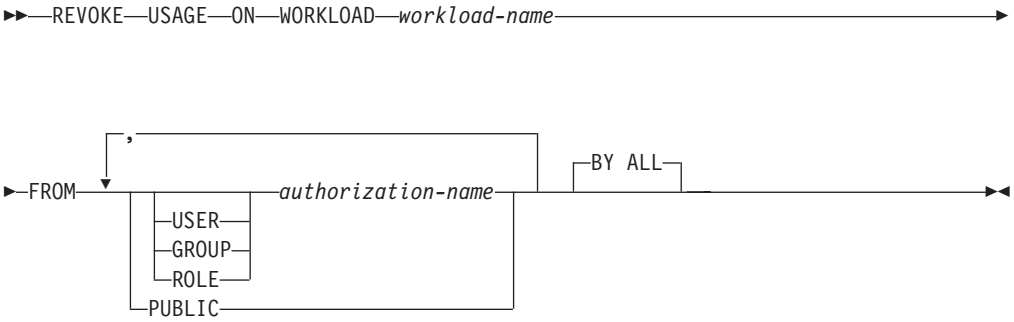
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include ACCESSCTRL, SECADM, or WLMADM authority.

Syntax



Description

USAGE
Revokes the privilege to use a workload.

ON WORKLOAD *workload-name*
Identifies the workload on which the USAGE privilege is to be revoked. This is a one-part name. The *workload-name* must identify a workload that exists at the current server (SQLSTATE 42704). The name cannot be 'SYSDEFAULTADMWORKLOAD' (SQLSTATE 42832).

FROM
Specifies from whom the USAGE privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

ROLE
Specifies that the *authorization-name* identifies an existing role at the current server (SQLSTATE 42704).

authorization-name,...
Lists the authorization IDs of one or more users, groups, or roles. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the USAGE privilege from PUBLIC.

BY ALL
Revokes the USAGE privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules

- For each *authorization-name* specified, if none of the keywords USER, GROUP, or ROLE is specified, then for all rows for the specified object in the SYSCAT.WORKLOADAUTH catalog view where the grantee is *authorization-name*:
  - If GRANTEETYPE is 'U', USER is assumed.
  - If GRANTEETYPE is 'G', GROUP is assumed.
  - If GRANTEETYPE is 'R', ROLE is assumed.
  - If GRANTEETYPE does not have the same value, an error is returned (SQLSTATE 56092.

Notes

- The REVOKE statement does not take effect until it is committed, even for the connection that issues the statement.

Example

Revoke the privilege to use the workload CAMPAIGN from user LISA.

```
REVOKE USAGE ON WORKLOAD CAMPAIGN FROM USER LISA
```

REVOKE (XSR object privileges)

This form of the REVOKE statement revokes USAGE privilege on an XSR object.

Invocation

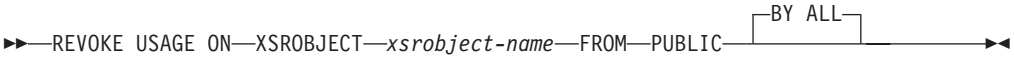
The REVOKE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if the DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

One of the following authorities is required:

- ACCESSCTRL or SECADM authority

Syntax



Description

ON XSROBJECT *xsrobject-name*
This name identifies the XSR object for which the USAGE privilege is revoked. The *xsrobject-name*, including the implicit or explicit schema qualifier, must uniquely identify an existing XSR object at the current server. If no XSR object by this name exists in the specified schema, an error is raised (SQLSTATE 42704).

FROM PUBLIC
Revokes the USAGE privilege from PUBLIC.

BY ALL
Revokes each named privilege from all users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Example

Revoke usage privileges on the XML schema MYSCHEMA from PUBLIC:

```
REVOKE USAGE ON XSROBJECT MYSCHEMA FROM PUBLIC
```

ROLLBACK

The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

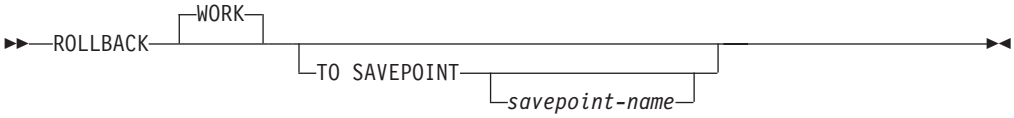
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

The unit of work in which the ROLLBACK statement is executed is terminated and a new unit of work is initiated. All changes made to the database during the unit of work are backed out.

The following statements, however, are not under transaction control, and changes made by them are independent of the ROLLBACK statement:

- SET CONNECTION
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE
- SET PASSTHRU (Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.)
- SET SERVER OPTION
- A SET statement that sets an updatable special register

The generation of sequence and identity values is not under transaction control. Values generated and consumed by the *nextval-expression* or by inserting rows into a table that has an identity column are independent of issuing the ROLLBACK statement. Also, issuing the ROLLBACK statement does not affect the value returned by the *prevval-expression*, nor the IDENTITY_VAL_LOCAL function.

Modification of the values of global variables is not under transaction control. ROLLBACK statements do not affect the values assigned to global variables.

TO SAVEPOINT
Specifies that a partial rollback (ROLLBACK TO SAVEPOINT) is to be performed. If no savepoint is active in the current savepoint level (see the "Rules" section in the description of the SAVEPOINT statement), an error is returned (SQLSTATE 3B502). After a successful rollback, the savepoint continues to exist, but any nested savepoints are released and no longer exist.

The nested savepoints, if any, are considered to have been rolled back and then released as part of the rollback to the current savepoint. If a *savepoint-name* is not provided, rollback occurs to the most recently set savepoint within the current savepoint level.

If this clause is omitted, the ROLLBACK statement rolls back the entire transaction. Furthermore, savepoints within the transaction are released.

savepoint-name
Specifies the savepoint that is to be used in the rollback operation. The specified *savepoint-name* cannot begin with 'SYS' (SQLSTATE 42939). After a successful rollback operation, the named savepoint continues to exist. If the savepoint name does not exist, an error (SQLSTATE 3B001) is returned. Data and schema changes made since the savepoint was set are undone.

Notes

- All locks held are released on a ROLLBACK of the unit of work. All open cursors are closed. All LOB locators are freed.
- Executing a ROLLBACK statement does not affect either the SET statements that change special register values or the RELEASE statement.
- If the program terminates abnormally, the unit of work is implicitly rolled back.
- Statement caching is affected by the rollback operation.
- The impact on cursors resulting from a ROLLBACK TO SAVEPOINT depends on the statements within the savepoint
  - If the savepoint contains DDL on which a cursor is dependent, the cursor is marked invalid. Attempts to use such a cursor results in an error (SQLSTATE 57007).
  - Otherwise:
    - If the cursor is referenced in the savepoint, the cursor remains open and is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.)
    - Otherwise, the cursor is not affected by the ROLLBACK TO SAVEPOINT (it remains open and positioned).
- Dynamic SQL statements prepared in a package bound with the `KEEPDYNAMIC YES` option are kept in the SQL context after a ROLLBACK statement. The statement might be implicitly prepared again, as a result of DDL operations that are rolled back within the unit of work.
- Inactive dynamic SQL statements prepared in a package bound with `KEEPDYNAMIC NO` are removed from the SQL context after a rollback operation. The statement must be prepared again before it can be executed in a new transaction.
- The following dynamic SQL statements may be active during ROLLBACK:
  - ROLLBACK statement
  - CALL statements under which the ROLLBACK statement was executed
- A ROLLBACK TO SAVEPOINT operation will drop any created temporary tables created within the savepoint. If a created temporary table is modified within the savepoint and that table has been defined as not logged, then all rows in the table are deleted.
- A ROLLBACK TO SAVEPOINT operation will drop any declared temporary tables declared within the savepoint. If a declared temporary table is modified within the savepoint and that table has been defined as not logged, then all rows in the table are deleted.
- All locks are retained after a ROLLBACK TO SAVEPOINT statement.

- All LOB locators are preserved following a ROLLBACK TO SAVEPOINT operation.

Example

Delete the alterations made since the last commit point or rollback.

```
ROLLBACK WORK
```

SAVEPOINT

Use the SAVEPOINT statement to set a savepoint within a transaction.

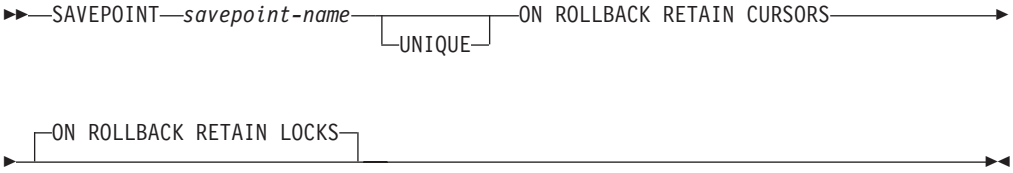
Invocation

This statement can be imbedded in an application program (including a procedure) or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

savepoint-name
Specifies the name of a savepoint. The specified *savepoint-name* cannot begin with 'SYS' (SQLSTATE 42939). If a savepoint by this name has already been defined as UNIQUE within this savepoint level, an error is returned (SQLSTATE 3B501).

UNIQUE
Specifies that the application does not intend to reuse this savepoint name while the savepoint is active within the current savepoint level. If *savepoint-name* already exists within this savepoint level, an error is returned (SQLSTATE 3B501).

ON ROLLBACK RETAIN CURSORS
Specifies system behavior upon rollback to this savepoint with respect to open cursor statements processed after the SAVEPOINT statement. This clause indicates that, whenever possible, the cursors are unaffected by a rollback to savepoint operation. For situations where the cursors are affected by the rollback to savepoint, see "ROLLBACK".

ON ROLLBACK RETAIN LOCKS
Specifies system behavior upon rollback to this savepoint with respect to locks acquired after the setting of the savepoint. Locks acquired since the savepoint are not tracked, and are not rolled back (released) upon rollback to the savepoint.

Rules

- Savepoint-related statements must not be used within trigger definitions (SQLSTATE 42987).
- A new savepoint level starts when one of the following events occurs:
  - A new unit of work (UOW) starts.
  - A procedure defined with the NEW SAVEPOINT LEVEL clause is called.

  - An atomic compound SQL statement starts.
- A savepoint level ends when the event that caused its creation is finished or removed. When a savepoint level ends, all savepoints contained within it are released. Any open cursors, DDL actions, or data modifications are inherited by the parent savepoint level (that is, the savepoint level within which the one that just ended was created), and are subject to any savepoint-related statements issued against the parent savepoint level.
- The following rules apply to actions within a savepoint level:
  - Savepoints can only be referenced within the savepoint level in which they are established. You cannot release, destroy, or roll back to a savepoint established outside of the current savepoint level.
  - All active savepoints established within the current savepoint level are automatically released when the savepoint level ends.
  - The uniqueness of savepoint names is only enforced within the current savepoint level. The names of savepoints that are active in other savepoint levels can be reused in the current savepoint level without affecting those savepoints in other savepoint levels.

Notes

- Once a SAVEPOINT statement has been issued, insert, update, or delete operations on nicknames are not allowed.
- Omitting the UNIQUE clause specifies that *savepoint-name* can be reused within the savepoint level by another savepoint. If a savepoint of the same name already exists within the savepoint level, the existing savepoint is destroyed and a new savepoint with the same name is created at the current point in processing. The new savepoint is considered to be the last savepoint established by the application. Note that the destruction of a savepoint through the reuse of its name by another savepoint simply destroys that one savepoint and does not release any savepoints established after the destroyed savepoint. These subsequent savepoints can only be released by means of the RELEASE SAVEPOINT statement, which releases the named savepoint and all savepoints established after the named savepoint.
- If the UNIQUE clause is specified, *savepoint-name* can only be reused after an existing savepoint with the same name has been released.
- Within a savepoint, if a utility, SQL statement, or DB2 command performs intermittent commits during processing, the savepoint will be implicitly released.
- If the SET INTEGRITY statement is rolled back within the savepoint, dynamically prepared statement names are still valid, although the statement might be implicitly prepared again.
- If inserts are buffered (that is, the application was precompiled with the INSERT BUF option), the buffer will be flushed when SAVEPOINT, ROLLBACK, or RELEASE TO SAVEPOINT statements are issued.

Example

Perform a rollback operation for nested savepoints. First, create a table named DEPARTMENT. Insert a row before starting SAVEPOINT1; insert another row and start SAVEPOINT2; then, insert a third row and start SAVEPOINT3.

```
CREATE TABLE DEPARTMENT (
  DEPTNO   CHAR(6),
  DEPTNAME VARCHAR(20),
  MGRNO    INTEGER)

  INSERT INTO DEPARTMENT VALUES ('A20', 'MARKETING', 301)
```

```
SAVEPOINT SAVEPOINT1 ON ROLLBACK RETAIN CURSORS

INSERT INTO DEPARTMENT VALUES ('B30', 'FINANCE', 520)

SAVEPOINT SAVEPOINT2 ON ROLLBACK RETAIN CURSORS

INSERT INTO DEPARTMENT VALUES ('C40', 'IT SUPPORT', 430)

SAVEPOINT SAVEPOINT3 ON ROLLBACK RETAIN CURSORS

INSERT INTO DEPARTMENT VALUES ('R50', 'RESEARCH', 150)
```

At this point, the DEPARTMENT table exists with rows A20, B30, C40, and R50. If you now issue:

```
ROLLBACK TO SAVEPOINT SAVEPOINT3
```

row R50 is no longer in the DEPARTMENT table. If you then issue:

```
ROLLBACK TO SAVEPOINT SAVEPOINT1
```

the DEPARTMENT table still exists, but the rows inserted since SAVEPOINT1 was established (B30 and C40) are no longer in the table.

SELECT

The SELECT statement is a form of query

The SELECT statement can be embedded in an application program or issued interactively.

SELECT INTO

The SELECT INTO statement produces a result table consisting of at most one row, and assigns the values in that row to host variables.

If the table is empty, the statement assigns +100 to SQLCODE and '02000' to SQLSTATE and does not assign values to the host variables. If more than one row satisfies the search condition, statement processing is terminated, and an error occurs (SQLSTATE 21000).

Invocation

This statement can be embedded only in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- SELECT privilege on the table, view, or nickname
- CONTROL privilege on the table, view, or nickname
- DATAACCESS authority

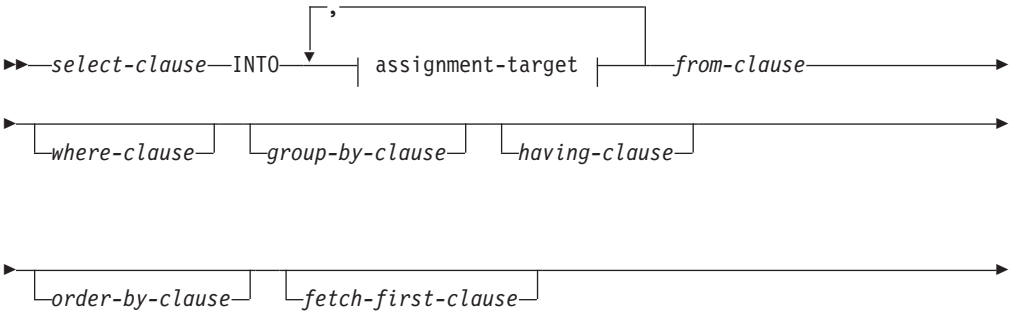
For each global variable used as an assignment target, the privileges held by the authorization ID of the statement must include one of the following authorities:

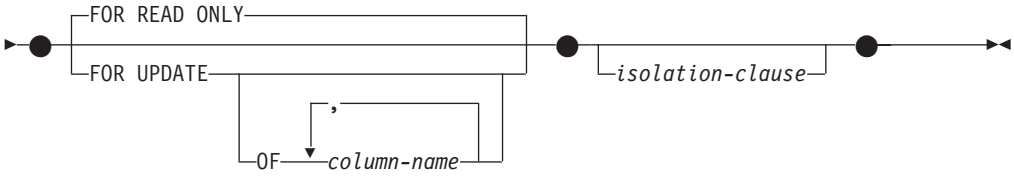
- WRITE privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

GROUP privileges are not checked for static SELECT INTO statements.

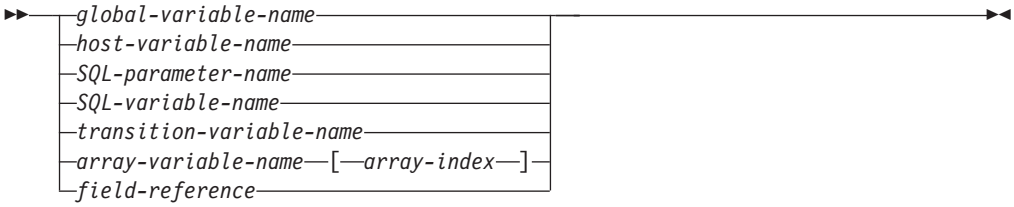
If the target of the SELECT INTO statement is a nickname, privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges that are required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax





assignment-target



Description

For a description of the *select-clause*, *from-clause*, *where-clause*, *group-by-clause*, *having-clause*, *order-by-clause*, *fetch-first-clause*, and *isolation-clause*, see "Queries" in the *SQL Reference Volume 1*.

INTO *assignment-target*
Identifies one or more targets for the assignment of output values.

The first value in the result row is assigned to the first target in the list, the second value to the second target, and so on. Each assignment to an *assignment-target* is made in sequence through the list. If an error occurs on any assignment, no value is assigned to any *assignment-target*.

When the data type of every *assignment-target* is not a row type, then the value 'W' is assigned to the SQLWARN3 field of the SQLCA if the number of *assignment-targets* is less than the number of result column values.

If the data type of an *assignment-target* is a row type, then there must be exactly one *assignment-target* specified (SQLSTATE 428HR), the number of columns must match the number of fields in the row type, and the data types of the columns of the fetched row must be assignable to the corresponding fields of the row type (SQLSTATE 42821).

If the data type of an *assignment-target* is an array element, then there must be exactly one *assignment-target* specified.

global-variable-name
Identifies the global variable that is the assignment target.

host-variable-name
Identifies the host variable that is the assignment target. For LOB output values, the target can be a regular host variable (if it is large enough), a LOB locator variable, or a LOB file reference variable.

SQL-parameter-name
Identifies the parameter that is the assignment target.

SQL-variable-name
Identifies the SQL variable that is the assignment target. SQL variables must be declared before they are used.

transition-variable-name
Identifies the column to be updated in the transition row. A *transition-variable-name* must identify a column in the subject table of a trigger, optionally qualified by a correlation name that identifies the new value.

array-variable-name
Identifies an SQL variable, SQL parameter, or global variable of an array type.

[array-index]
An expression that specifies which element in the array will be the target of the assignment. For an ordinary array, the *array-index* expression must be assignable to INTEGER (SQLSTATE 428H1) and cannot be the null value. Its value must be between 1 and the maximum cardinality defined for the array (SQLSTATE 2202E). For an associative array, the *array-index* expression must be assignable to the index data type of the associative array (SQLSTATE 428H1) and cannot be the null value.

field-reference
Identifies the field within a row type value that is the assignment target. The *field-reference* must be specified as a qualified *field-name* where the qualifier identifies the row value in which the field is defined.

FOR READ ONLY or FOR UPDATE
Indicates the intended use for the selected row. The default is FOR READ ONLY.

FOR READ ONLY
Specifies that the selected row will not be locked for update.

FOR UPDATE
Specifies that the selected row from the underlying table will be locked to facilitate updating the row later on in the transaction, similar to the locking done for the select statement of a cursor which includes the FOR UPDATE clause.

FOR UPDATE must not be specified if the result table of the SELECT INTO statement is read-only (SQLSTATE 42829).
If *column-name* values are listed, these columns must be updatable (SQLSTATE 42829).
Note that listing columns has only documentary effect and does not limit subsequent searched update statements from modifying other columns.

Rules

- Global variables cannot be assigned inside triggers that are not defined using a compound SQL (compiled) statement, functions that are not defined using a compound SQL (compiled) statement, methods, or compound SQL (inlined) statements (SQLSTATE 428GX).

Notes

- **Syntax alternatives:** For consistency with SQL queries:
  - FOR FETCH ONLY can be specified in place of FOR READ ONLY

Examples

- *Example 1:* This C example puts the maximum salary in the EMP table into the host variable MAXSALARY.

```
EXEC SQL SELECT MAX(SALARY)
  INTO :MAXSALARY
  FROM EMP;
```

- *Example 2:* This C example puts the row for employee 528671 (from the EMP table) into host variables.

```
EXEC SQL SELECT * INTO :h1, :h2, :h3, :h4
  FROM EMP
  WHERE EMPNO = '528671';
```

- *Example 3:* This SQLJ example puts the row for employee 528671 (from the EMP table) into host variables. That row will later be updated using a searched update, and should be locked when the query executes.

```
#sql { SELECT * INTO :FIRSTNAME, :LASTNAME, :EMPNO, :SALARY
  FROM EMP
  WHERE EMPNO = '528671'
  FOR UPDATE };
```

- *Example 4:* This C example puts the maximum salary in the EMP table into the global variable GV_MAXSALARY.

```
EXEC SQL SELECT MAX(SALARY)
  INTO GV_MAXSALARY
  FROM EMP;
```

SET COMPILATION ENVIRONMENT

The SET COMPILATION ENVIRONMENT statement changes the current compilation environment in the connection to match the values contained in the compilation environment provided by an event monitor. This statement changes the values of one or more special registers; these changes, in turn, will affect the compilation of any subsequent dynamic SQL statement.

This statement is not under transaction control.

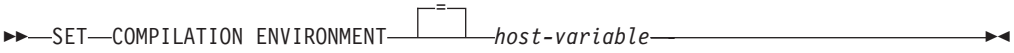
Invocation

The statement can be embedded in an application program. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

host-variable
A variable of type BLOB containing a compilation environment provided by an event monitor. It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815). If the format of the compilation environment is incorrect, an error is returned, and the connection settings remain unmodified (SQLSTATE 51040).

Notes

- To reset the compilation environment to the original default values, terminate and then restart the connection. You can achieve the same effect by issuing this statement within an SQL routine, so that any special register changes are not reflected in the connection upon return from that routine.
- Use the COMPILATION_ENV table function to look at the individual elements that are contained within the compilation environment.

Example

Set the current session's compilation environment to the values contained in a compilation environment that was previously captured by a deadlock event monitor. A deadlock event monitor that is created specifying the WITH DETAILS HISTORY option will capture the compilation environment for dynamic SQL statements. This captured environment is what is accepted as input to the statement.

```
SET COMPILATION ENVIRONMENT = :hv1
```

SET CONNECTION

The SET CONNECTION statement changes the state of a connection from dormant to current, making the specified location the current server.

This statement is not under transaction control.

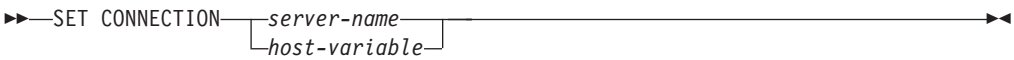
Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax



Description

server-name **or** *host-variable*

Identifies the application server by the specified *server-name* or a *host-variable* which contains the *server-name*.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-aligned and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

The *server-name* or the *host-variable* must identify an existing connection of the application process. If they do not identify an existing connection, an error (SQLSTATE 08003) is raised.

If SET CONNECTION is to the current connection, the states of all connections of the application process are unchanged.

Successful Connection

If the SET CONNECTION statement executes successfully:

- No connection is made. The CURRENT SERVER special register is updated with the specified *server-name*.
- The previously current connection, if any, is placed into the dormant state (assuming a different *server-name* is specified).
- The CURRENT SERVER special register and the SQLCA are updated in the same way as documented under “CONNECT (Type 1)”.

Unsuccessful Connection

If the SET CONNECTION statement fails:

- No matter what the reason for failure, the connection state of the application process and the states of its connections are unchanged.

- As with an unsuccessful Type 1 CONNECT, the SQLERRP field of the SQLCA is set to the name of the module that detected the error.

Notes

- The use of type 1 CONNECT statements does not preclude the use of SET CONNECTION, but the statement will always fail (SQLSTATE 08003), unless the SET CONNECTION statement specifies the current connection, because dormant connections cannot exist.
- The SQLRULES(DB2) connection option (see "Options that Govern Distributed Unit of Work Semantics") does not preclude the use of SET CONNECTION, but the statement is unnecessary, because type 2 CONNECT statements can be used instead.
- When a connection is used, made dormant, and then restored to the current state in the same unit of work, that connection reflects its last use by the application process with regard to the status of locks, cursors, and prepared statements.

Example

Execute SQL statements at IBMSTHDB, execute SQL statements at IBMTOKDB, and then execute more SQL statements at IBMSTHDB.

```
EXEC SQL CONNECT TO IBMSTHDB;
/* Execute statements referencing objects at IBMSTHDB */
EXEC SQL CONNECT TO IBMTOKDB;
/* Execute statements referencing objects at IBMTOKDB */
EXEC SQL SET CONNECTION IBMSTHDB;
/* Execute statements referencing objects at IBMSTHDB */
```

Note that the first CONNECT statement creates the IBMSTHDB connection, the second CONNECT statement places it in the dormant state, and the SET CONNECTION statement returns it to the current state.

SET CURRENT DECFLOAT ROUNDING MODE

The SET CURRENT DECFLOAT ROUNDING MODE statement verifies that the specified rounding mode is the value that is currently set for the CURRENT DECFLOAT ROUNDING MODE special register.

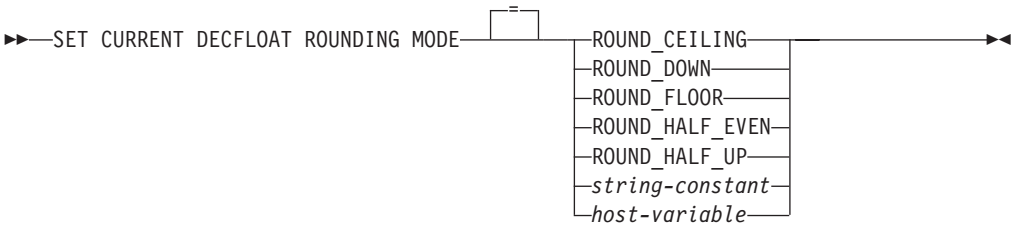
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

ROUND_CEILING
Round the value toward positive infinity. If all of the discarded digits are zero or if the sign is negative, the result is unchanged (except for the removal of the discarded digits). Otherwise, the result coefficient is incremented by 1.

ROUND_DOWN
Round the value toward 0 (truncation). The discarded digits are ignored.

ROUND_FLOOR
Round the value toward negative infinity. If all of the discarded digits are zero or if the sign is positive, the result is unchanged (except for the removal of the discarded digits). Otherwise, the sign is negative and the result coefficient is incremented by 1.

ROUND_HALF_EVEN
Round the value to the nearest value. If the values are equidistant, round the value so that the final digit is even. If the discarded digits represent more than half of the value of a number in the next left position, the result coefficient is incremented by 1. If they represent less than half, the result coefficient is not adjusted (that is, the discarded digits are ignored). Otherwise, the result coefficient is unaltered if its rightmost digit is even, or incremented by 1 if its rightmost digit is odd (to make an even digit).

ROUND_HALF_UP
Round the value to the nearest value. If the values are equidistant, round the value up. If the discarded digits represent half or more than half of the value of a number in the next left position, the result coefficient is incremented by 1. Otherwise, the discarded digits are ignored.

string-constant
A character string constant with a maximum length of 15 bytes, after trailing blanks have been removed. The value must be a left-aligned string that specifies one of the five rounding mode keywords (case insensitive).

host-variable
A variable of type CHAR or VARCHAR. The value of the host variable must be a left-aligned string that specifies one of the five rounding mode keywords (case insensitive). The actual length of the contents of *host-variable* must not be greater than 15 bytes, after trailing blanks have been removed. The value must be padded on the right with blanks when using a fixed-length character host variable. The host variable cannot be set to the null value.

Rules

- The specified rounding mode value must be the same as the value of the CURRENT DECFLOAT ROUNDING MODE special register (SQLSTATE 42815).

Notes

- This statement does not change the value of the CURRENT DECFLOAT ROUNDING MODE special register on a DB2 for Linux, UNIX, and Windows server. However, when the statement is processed by a DB2 for z/OS server or a DB2 for i server, it can be used to change the value of the CURRENT DECFLOAT ROUNDING MODE special register on that server.

Example

The following statement verifies whether the specified rounding mode value for the client matches the rounding mode value that is currently set on the server.

```
SET CURRENT DECFLOAT ROUNDING MODE = ROUND_CEILING
```

SET CURRENT DEFAULT TRANSFORM GROUP

The SET CURRENT DEFAULT TRANSFORM GROUP statement changes the value of the CURRENT DEFAULT TRANSFORM GROUP special register.

This statement is not under transaction control.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax

Description

group-name
Specifies a one-part name that identifies a transform group defined for all structured types. This name can be referenced in subsequent statements (or until the special register value is changed again using another SET CURRENT DEFAULT TRANSFORM GROUP statement).

The name must be an SQL identifier, up to 128 bytes in length (SQLSTATE 42815). No validation that the *group-name* is defined for any structured type is made when the special register is set. Only when a structured type is specifically referenced is the definition of the named transform group checked for validity.

Rules

- If the value specified does not conform to the rules for a *group-name*, an error is raised (SQLSTATE 42815)
- The TO SQL and FROM SQL functions defined in the *group-name* transform group are used for exchanging user-defined structured type data with a host program.

Notes

- The initial value of the CURRENT DEFAULT TRANSFORM GROUP special register is the empty string.

Example

Set the default transform group to MYSTRUCT1. The TO SQL and FROM SQL functions defined in the MYSTRUCT1 transform group will be used for exchanging user-defined structured type variables with the current host program.

```
SET CURRENT DEFAULT TRANSFORM GROUP = MYSTRUCT1
```

SET CURRENT DEGREE

The SET CURRENT DEGREE statement assigns a value to the CURRENT DEGREE special register.

This statement is not under transaction control.

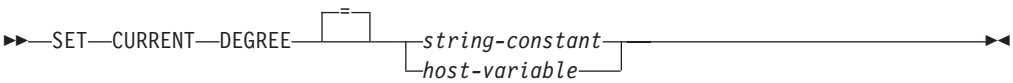
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

The value of CURRENT DEGREE is replaced by the value of the string constant or host variable. The value must be a character string that is not longer than 5 bytes. The value must be the character string representation of an integer between 1 and 32 767 inclusive or 'ANY'.

If the value of CURRENT DEGREE represented as an integer is 1 when an SQL statement is dynamically prepared, the execution of that statement will not use intrapartition parallelism.

If the value of CURRENT DEGREE is a number when an SQL statement is dynamically prepared, the execution of that statement can involve intrapartition parallelism with the specified degree.

If the value of CURRENT DEGREE is 'ANY' when an SQL statement is dynamically prepared, the execution of that statement can involve intrapartition parallelism using a degree determined by the database manager.

host-variable
The *host-variable* must be of data type CHAR or VARCHAR and the length must not exceed 5. If a longer field is provided, an error will be returned (SQLSTATE 42815). If the actual value provided is larger than the replacement value specified, the input must be padded on the right with blanks. Leading blanks are not allowed (SQLSTATE 42815). All input values are treated as being case-insensitive. If a *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

string-constant
The *string-constant* length must not exceed 5.

Notes

- The degree of intrapartition parallelism for static SQL statements can be controlled using the DEGREE option of the PREP or BIND command.
- The actual runtime degree of intrapartition parallelism will be the lower of:
  - Maximum query degree (**max_querydegree**) configuration parameter
  - Application runtime degree
  - SQL statement compilation degree
- The **intra_parallel** database manager configuration parameter must be on to use intrapartition parallelism. If it is set to off, the value of this register will be ignored and the statement will not use intrapartition parallelism for the purpose of optimization (SQLSTATE 01623).
- The value in the CURRENT DEGREE special register and the **intra_parallel** setting can be overridden in a workload by setting the MAXIMUM DEGREE workload attribute.
- Some SQL statements cannot use intrapartition parallelism.

Examples

- *Example 1:* The following statement sets the CURRENT DEGREE to inhibit intrapartition parallelism.

```
SET CURRENT DEGREE = '1'
```

- *Example 2:* The following statement sets the CURRENT DEGREE to allow intrapartition parallelism.

```
SET CURRENT DEGREE = 'ANY'
```

SET CURRENT EXPLAIN MODE

The SET CURRENT EXPLAIN MODE statement changes the value of the CURRENT EXPLAIN MODE special register. It is not under transaction control.

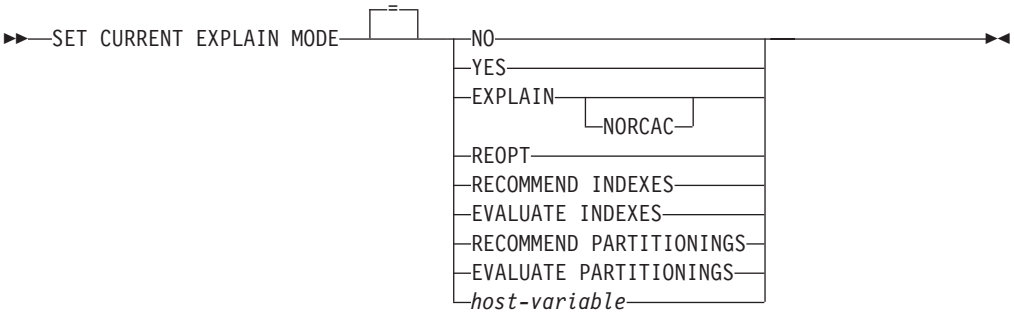
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

NO Disables the Explain facility. No Explain information is captured. NO is the initial value of the special register.

YES
Enables the Explain facility and causes Explain information to be inserted into the Explain tables for eligible dynamic SQL statements. All dynamic SQL statements are compiled and executed normally.

EXPLAIN
Enables the Explain facility and causes Explain information to be captured for any eligible dynamic SQL statement that is prepared. However, dynamic statements are not executed.

EXPLAIN NORCAC
Enables the Explain facility and causes Explain information to be captured for any eligible dynamic SQL statement that is prepared as if row or column access control (RCAC) was not activated. Dynamic statements are not executed. When this explain mode is set, explain facility would explain the plan as if RCAC was not present.

REOPT
Enables the Explain facility and causes Explain information to be captured for a static or dynamic SQL statement during statement reoptimization at execution time; that is, when actual values for the host variables, special registers, global variables, or parameter markers are available.

RECOMMEND INDEXES
Enables the SQL compiler to recommend indexes. All queries that are executed in this explain mode will populate the ADVISE_INDEX table with

recommended indexes. In addition, Explain information will be captured in the Explain tables to reveal how the recommended indexes are used, but the statements are neither compiled nor executed.

EVALUATE INDEXES
Enables the SQL compiler to evaluate virtual recommended indexes for dynamic queries. Queries executed in this explain mode will be compiled and optimized using fabricated statistics based on the virtual indexes. The statements are not executed. The indexes to be evaluated are read from the ADVISE_INDEX table if the USE_INDEX column contains 'Y'. Existing non-unique indexes can also be ignored by setting the USE_INDEX column to 'I' and the EXISTS column to 'Y'. If a combination of USE_INDEX='I' and EXISTS='N' is given then index evaluation for the query will continue normally but the index in question will not be ignored.

RECOMMEND PARTITIONINGS
Specifies that the compiler is to recommend the best database partition for each table that is accessed by a specific query. The best database partitions are then written to an ADVISE_PARTITION table. The query is not executed.

EVALUATE PARTITIONINGS
Specifies that the compiler is to obtain the estimated performance of a query using the virtual database partitions specified in the ADVISE_PARTITION table.

host-variable
The *host-variable* must be of data type CHAR or VARCHAR and the length must not exceed 254. If a longer field is provided, an error will be returned (SQLSTATE 42815). The value specified must be NO, YES, EXPLAIN, RECOMMEND INDEXES, or EVALUATE INDEXES. If the actual value provided is larger than the replacement value specified, the input must be padded on the right with blanks. Leading blanks are not allowed (SQLSTATE 42815). All input values are treated as being case-insensitive. If a *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

Notes

- The Explain facility uses the following IDs as the schema when qualifying Explain tables that it is populating:
  - The session authorization ID for dynamic SQL
  - The statement authorization ID for static SQL

  The schema can be associated with a set of Explain tables, or aliases that point to a set of Explain tables under a different schema. If no Explain tables are found under the schema, the Explain facility checks for Explain tables under the SYSTOOLS schema and attempts to use those tables.
- Explain information for static SQL statements can be captured by using the **EXPLAIN** option of the **PREP** or **BIND** command. If the ALL value of the **EXPLAIN** option is specified, and the CURRENT EXPLAIN MODE register value is NO, explain information will be captured for dynamic SQL statements at run time. If the value of the CURRENT EXPLAIN MODE register is not NO, the value of the **EXPLAIN** bind option is ignored.
- RECOMMEND INDEXES and EVALUATE INDEXES are special modes which can only be set with the SET CURRENT EXPLAIN MODE statement. These modes cannot be set using **PREP** or **BIND** options, and they do not work with the SET CURRENT EXPLAIN SNAPSHOT statement.

- If the Explain facility is activated, the current authorization ID must have INSERT privilege for the Explain tables, or an error (SQLSTATE 42501) is raised.
- When SQL statements are explained from a routine, the routine must be defined with an SQL data access indicator of MODIFIES SQL DATA (SQLSTATE 42985).
- If the special register is set to REOPT, and the SQL statement does not qualify for reoptimization at execution time (that is, if the statement does not have input variables, or if the **REOPT** bind option is set to NONE), then no Explain information will be captured. If the **REOPT** bind option is set to ONCE, Explain information will be captured only once when the statement is initially reoptimized. After the statement is cached, no further Explain information will be acquired for this statement on subsequent executions.
- If the Explain facility is enabled, the **REOPT** bind option is set to ONCE, and you attempt to execute an SQL statement that is already cached, the statement will be compiled and reoptimized with the current values of the input variables, and the Explain tables will be populated accordingly. The newly generated access plan for this statement will not be cached or executed. Other applications that are concurrently executing this cached statement will continue to execute, and new requests to execute this statement will pick up the already cached access plan.
- A value of REOPT for the CURRENT EXPLAIN MODE and CURRENT EXPLAIN SNAPSHOT special registers will override the value of the **EXPLAIN** and **EXPLSNAP** bind options at bind time if a static or dynamic SQL statement has input variables, and the **REOPT** bind option is set to ONCE or ALWAYS.
- Row and column level access control (RCAC) defined on the EXPLAIN tables is enforced for user access to these tables just like any other regular tables. However, row and column level access control on the EXPLAIN tables is not enforced when the DB2 database itself is populating those EXPLAIN tables. This is considered internal housekeeping and is not subject to RCAC, much like internal SQL.

Example

The following statement sets the CURRENT EXPLAIN MODE special register, so that Explain information will be captured for any subsequent eligible dynamic SQL statements and the statement will not be executed.

```
SET CURRENT EXPLAIN MODE = EXPLAIN
```

SET CURRENT EXPLAIN SNAPSHOT

The SET CURRENT EXPLAIN SNAPSHOT statement changes the value of the CURRENT EXPLAIN SNAPSHOT special register.

This statement is not under transaction control.

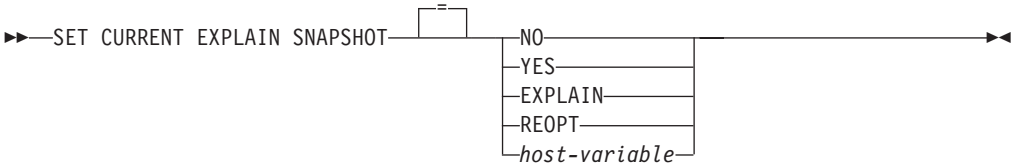
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

NO Disables the Explain snapshot facility. No snapshot is taken. NO is the initial value of the special register.

YES
Enables the Explain snapshot facility, creating a snapshot of the internal representation for each eligible dynamic SQL statement. This information is inserted in the SNAPSHOT column of the EXPLAIN_STATEMENT table.

EXPLAIN
Enables the Explain snapshot facility, creating a snapshot of the internal representation for each eligible dynamic SQL statement that is prepared. However, dynamic statements are not executed.

REOPT
Enables the Explain facility and causes Explain information to be captured for a static or dynamic SQL statement during statement reoptimization at execution time; that is, when actual values for the host variables, special registers, global variables, or parameter markers are available.

host-variable
The *host-variable* must be of data type CHAR or VARCHAR and the length of its contents must not exceed 8. If a longer field is provided, an error will be returned (SQLSTATE 42815). The value contained in this register must be either NO, YES, or EXPLAIN. If the actual value provided is larger than the replacement value specified, the input must be padded on the right with blanks. Leading blanks are not allowed (SQLSTATE 42815). All input values are treated as being case-insensitive. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

Notes

- The Explain facility uses the following IDs as the schema when qualifying Explain tables that it is populating:
  - The session authorization ID for dynamic SQL
  - The statement authorization ID for static SQL

  The schema can be associated with a set of Explain tables, or aliases that point to a set of Explain tables under a different schema. If no Explain tables are found under the schema, the Explain facility checks for Explain tables under the SYSTOOLS schema and attempts to use those tables.
- Explain snapshots for static SQL statements can be captured by using the EXPLSNAP option of the PREP or BIND command. If the ALL value of the EXPLSNAP option is specified, and the CURRENT EXPLAIN SNAPSHOT register value is NO, Explain snapshots will be captured for dynamic SQL statements at run time. If the value of the CURRENT EXPLAIN SNAPSHOT register is not NO, the EXPLSNAP option is ignored.
- If the Explain snapshot facility is activated, the current authorization ID must have INSERT privilege for the Explain tables or an error (SQLSTATE 42501) is raised.
- When SQL statements are explained from a routine, the routine must be defined with an SQL data access indicator of MODIFIES SQL DATA (SQLSTATE 42985).
- If the special register is set to REOPT, and the SQL statement does not qualify for reoptimization at execution time (that is, if the statement does not have input variables, or if the REOPT bind option is set to NONE), then no Explain information will be captured. If the REOPT bind option is set to ONCE, Explain snapshot information will be captured only once when the statement is initially reoptimized. After the statement is cached, no further Explain information will be acquired for this statement on subsequent executions.
- If the Explain facility is enabled, the REOPT bind option is set to ONCE, and you attempt to execute a reoptimizable SQL statement that is already cached, the statement will be compiled and reoptimized with the current values of the input variables, and the Explain snapshot will be captured accordingly. The newly generated access plan for this statement will not be cached or executed. Other applications that are concurrently executing this cached statement will continue to execute, and new requests to execute this statement will pick up the already cached access plan.
- The value REOPT for the CURRENT EXPLAIN MODE and CURRENT EXPLAIN SNAPSHOT special registers will override the value of the EXPLAIN and EXPLSNAP bind options at bind time if a static or dynamic SQL statement has input variables, and the REOPT bind option is set to ONCE or ALWAYS.

Examples

- *Example 1:* The following statement sets the CURRENT EXPLAIN SNAPSHOT special register, so that an Explain snapshot will be taken for any subsequent eligible dynamic SQL statements and the statement will be executed.

  ```
  SET CURRENT EXPLAIN SNAPSHOT = YES
  ```

- *Example 2:* The following example retrieves the current value of the CURRENT EXPLAIN SNAPSHOT special register into the host variable called SNAP.

  ```
  EXEC SQL VALUES (CURRENT EXPLAIN SNAPSHOT) INTO :SNAP;
  ```

SET CURRENT FEDERATED ASYNCHRONY

The SET CURRENT FEDERATED ASYNCHRONY statement assigns a value to the CURRENT FEDERATED ASYNCHRONY special register.

This statement is not under transaction control.

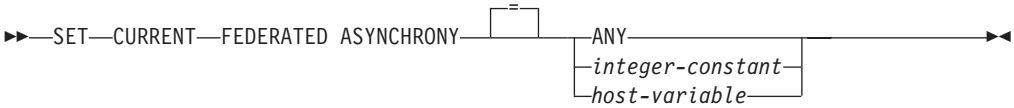
Invocation

The statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

ANY
Specifies a CURRENT FEDERATED ASYNCHRONY value of -1, which means that the execution of statements can involve asynchrony using a degree that is determined by the database manager.

integer-constant
Specifies an integer value between 0 and 32 767, inclusive. The execution of statements can involve asynchrony using the specified degree. If the value is 0 when an SQL statement is dynamically prepared, the execution of that statement will not use asynchrony.

host-variable
A variable of type INTEGER. The value must be between 0 and 32 767, inclusive, or -1 (representing ANY). If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

Notes

- The degree of asynchrony for static SQL statements can be controlled using the FEDERATED_ASYNCHRONY option of the PREP or BIND command.
- The initial value of the CURRENT FEDERATED ASYNCHRONY special register is determined by the **federated_async** database manager configuration parameter if the dynamic statement is issued through the command line processor (CLP). The initial value is determined by the FEDERATED_ASYNCHRONY bind option if the dynamic statement is part of an application that is being bound.

Examples

- *Example 1:* The following statement disables asynchrony by setting the value of the CURRENT FEDERATED ASYNCHRONY special register to 0.

```
SET CURRENT FEDERATED ASYNCHRONY = 0
```

- *Example 2:* The following statement sets the degree of asynchrony to 5.

```
SET CURRENT FEDERATED ASYNCHRONY 5
```

- *Example 3:* The following statement sets the value of the CURRENT FEDERATED ASYNCHRONY special register to -1, which specifies that the database manager is to determine the degree of asynchrony.

```
SET CURRENT FEDERATED ASYNCHRONY ANY
```

SET CURRENT IMPLICIT XMLPARSE OPTION

The SET CURRENT IMPLICIT XMLPARSE OPTION statement changes the value of the CURRENT IMPLICIT XMLPARSE OPTION special register.

This statement is not under transaction control.

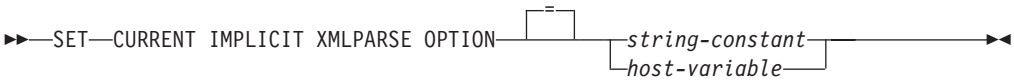
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

string-constant
A character string constant. The value must be a left-aligned string that is either 'PRESERVE WHITESPACE' or 'STRIP WHITESPACE' (case insensitive) with no additional blank characters between the keywords.

host-variable
A variable of type CHAR or VARCHAR. The value of the host variable must be a left-aligned string that is either 'PRESERVE WHITESPACE' or 'STRIP WHITESPACE' (case insensitive) with no additional blank characters between the keywords. The value must be padded on the right with blanks when using a fixed-length character *host-variable*. The host variable cannot be set to null.

Notes

- The initial value of the CURRENT IMPLICIT XMLPARSE OPTION special register is 'STRIP WHITESPACE'.
- Both dynamic and static SQL statements are affected by this special register.

Example

Set the value of the CURRENT IMPLICIT XMLPARSE OPTION special register to 'PRESERVE WHITESPACE'.

```
SET CURRENT IMPLICIT XMLPARSE OPTION = 'PRESERVE WHITESPACE'
```

SET CURRENT ISOLATION

The SET CURRENT ISOLATION statement assigns a value to the CURRENT ISOLATION special register.

This statement is not under transaction control.

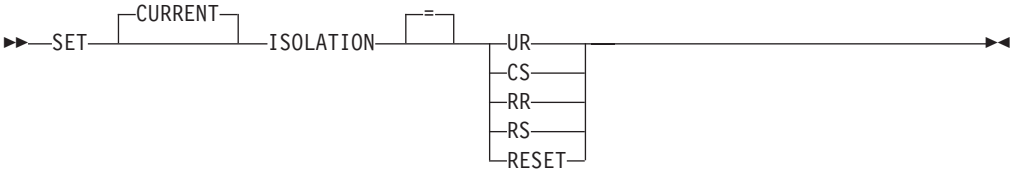
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

The value of the CURRENT ISOLATION special register is replaced by the specified value or set to blanks if RESET is specified.

Notes

- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products.
  - TO can be specified in place of the equal sign (=)
  - DIRTY READ can be specified in place of UR
  - READ UNCOMMITTED can be specified in place of UR
  - READ COMMITTED is recognized and upgraded to CS
  - CURSOR STABILITY can be specified in place of CS
  - REPEATABLE READ can be specified in place of RR
  - SERIALIZABLE can be specified in place of RR

SET CURRENT LOCALE LC_MESSAGES

The SET CURRENT LOCALE LC_MESSAGES statement changes the value of the CURRENT LOCALE LC_MESSAGES special register.

This statement is not under transaction control.

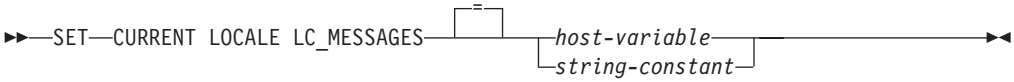
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

The CURRENT LOCALE LC_MESSAGES special register identifies the locale that is used by EVMON_UPGRADE_TABLES, as well as monitoring routines in the **monreport** module. EVMON_UPGRADE_TABLES and the monitoring routines use the value of CURRENT LOCALE LC_MESSAGES to determine in which language the result set text output should be returned. User-defined routines that are coded to return messages could also use the value of CURRENT LOCALE LC_MESSAGES to determine what language to use for message text.

host-variable
A variable of type CHAR or VARCHAR. It cannot be set to null.

string-constant
A character string constant.

Notes

- ***Initial value:*** The initial value of the CURRENT LOCALE LC_MESSAGES special register is 'en_US'.
- ***Language availability:*** If the language for the locale is not available to the DB2 database manager, messages will be returned in English.
- ***Code page compatibility:*** The language for the locale specified must be supported by the code page of the output parameter or returns type of a routine that uses the special register to determine what language to return message text information in. If the database is not a Unicode database (and the routine was not created with PARAMETER CCSID UNICODE) and some characters in the language for the locale cannot be represented in the database code page, substitution characters will be returned as a result of code page conversion.
- ***Potential future use:*** In a future release, the value of the CURRENT LOCALE LC_MESSAGES special register might be used for other areas of the database environment that involve messages.
- ***Valid locales and naming:*** For information about valid locales and their naming, see "Locale names for SQL and XQuery" in the *Globalization Guide*

Examples

- *Example 1:* The following statement sets the CURRENT LOCALE LC_MESSAGES special register to the English (Canada) locale using the latest version of Common Locale Data Repository (CLDR) available in DB2 database manager.

```
SET CURRENT LOCALE LC_MESSAGES = 'en_CA'
```

- *Example 2:* The following statement sets the CURRENT LOCALE LC_MESSAGES special register to the French (France) locale using Common Locale Data Repository (CLDR) version 1.5. The CONNECTION routine in the **monreport** module is then invoked to have its output returned in French.

```
SET CURRENT LOCALE LC_MESSAGES = 'CLDR 1.5:fr_FR'
CALL MONREPORT.CONNECTION
```

- *Example 3:* Assume that the user-defined procedure XYZ.STORELOCATOR takes a zip code or postal code input. It returns a result set of stores of the XYZ company within a 30 minute drive from the zip code or postal code given as input. If the zip code or postal code is not in the correct format, an error message is returned that indicates what the problem is with the format. The procedure is coded to be able to return the error message in the language determined from the value of the CURRENT LOCALE LC_MESSAGES special register. The following statement sets the CURRENT LOCALE LC_MESSAGES special register to the Spanish (Mexico) locale. The store locator user-defined procedure is then invoked and any error messages will be returned in Spanish.

```
SET CURRENT LOCALE LC_MESSAGES = 'es_MX'
CALL XYZ.STORELOCATOR(:ZIP, :STATUSMSG)
```

SET CURRENT LOCALE LC_TIME

The SET CURRENT LOCALE LC_TIME statement changes the value of the CURRENT LOCALE LC_TIME special register. It is not under transaction control.

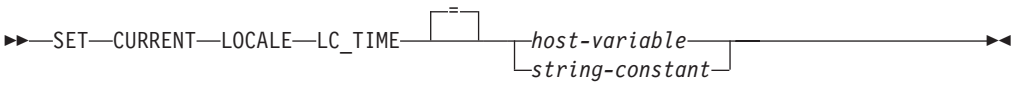
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

The CURRENT LOCALE LC_TIME special register is used by the DAYNAME, MONTHNAME, NEXT_DAY, ROUND, ROUND_TIMESTAMP, TIMESTAMP_FORMAT, TRUNCATE, TRUNC_TIMESTAMP and VARCHAR_FORMAT functions when the *locale-name* argument is not explicitly specified.

host-variable
A variable of type CHAR or VARCHAR. It cannot be set to null.

string-constant
A character string constant.

Notes

- ***Initial Value:*** The initial value of the CURRENT LOCALE LC_TIME special register is 'en_US'.
- ***Potential future use:*** In a future release the value of the CURRENT LOCALE LC_TIME special register might be used by other scalar functions and for other areas of the database environment that involve datetime values.
- ***Valid locales and naming:*** For information on valid locales and their naming,, see “Locale names for SQL and XQuery” in the *Globalization Guide* .

Examples

- *Example 1:* The following statement sets the CURRENT LOCALE LC_TIME special register to the English (Canada) locale using the latest version of Common Locale Data Repository (CLDR) available in DB2 database manager.

```
SET CURRENT LOCALE LC_TIME = 'en_CA'
```

- *Example 2:* The following statement sets the CURRENT LOCALE LC_TIME special register to the French (France) locale using Common Locale Data Repository (CLDR) version 1.8.1. The MONTHNAME scalar function is then invoked with a single argument of '2008-11-10-00.00.00.000000'.

```
SET CURRENT LOCALE LC_TIME = 'CLDR181_fr_FR'
VALUES MONTHNAME( '2008-11-10-00.00.00.000000' )
```

returns:

```
'novembre'
```

SET CURRENT LOCK TIMEOUT

The SET CURRENT LOCK TIMEOUT statement changes the value of the CURRENT LOCK TIMEOUT special register.

This statement is not under transaction control.

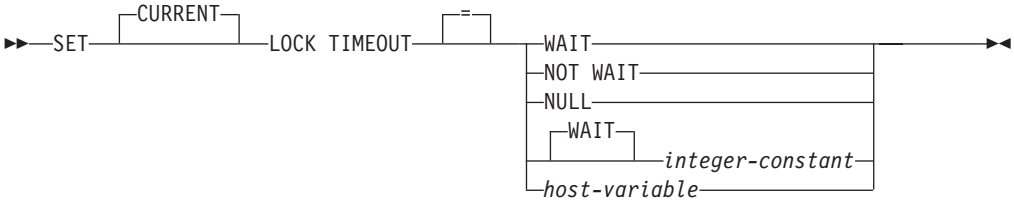
Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

The specified value must be an integer between -1 and 32767, inclusive (SQLSTATE 428B7), or the null value.

WAIT
Specifies a CURRENT LOCK TIMEOUT value of -1, which means that the database manager is to wait until a lock is released, or a deadlock is detected (SQLSTATE 40001 or 57033).

NOT WAIT
Specifies a CURRENT LOCK TIMEOUT value of 0, which means that the database manager is not to wait for locks that cannot be obtained, and an error (SQLSTATE 40001 or 57033) will be returned.

NULL
Specifies that the CURRENT LOCK TIMEOUT value is to be unset, and that the value of the **locktimeout** database configuration parameter is to be used when waiting for a lock. The value that is returned for the special register will change as the value of **locktimeout** changes.

WAIT *integer-constant*
Specifies an integer value between -1 and 32767. A value of -1 is equivalent to specifying the WAIT keyword without an integer value. A value of 0 is equivalent to specifying the NOT WAIT clause. If the value is between 1 and 32767, the database manager will wait that number of seconds (if a lock cannot be obtained) before an error (SQLSTATE 40001 or 57033) is returned.

host-variable
A variable of type INTEGER. The value must be between -1 and 32767. If *host-variable* has an associated indicator variable, and the value of that indicator

variable specifies a null value, the CURRENT LOCK TIMEOUT value is unset. This is equivalent to specifying the NULL keyword.

Notes

- An updated value of the special register takes effect immediately upon successful execution of this statement. Because the special register value that is to be used during statement execution is fixed at the beginning of statement execution, an updated value of the CURRENT LOCK TIMEOUT special register will only be returned by statements that start execution after the SET LOCK TIMEOUT statement has completed successfully.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with Informix database products. These alternatives are non-standard and should not be used.
  - MODE can be specified in place of TIMEOUT.
  - TO can be specified in place of the equals (=) operator.

Examples

- *Example 1:* Set the lock timeout value to wait for 30 seconds before returning an error.

```
SET CURRENT LOCK TIMEOUT 30
```

- *Example 2:* Unset the lock timeout value, so that the **locktimeout** database configuration parameter value will be used instead.

```
SET CURRENT LOCK TIMEOUT NULL
```

SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION

The SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION statement changes the value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.

This statement is not under transaction control.

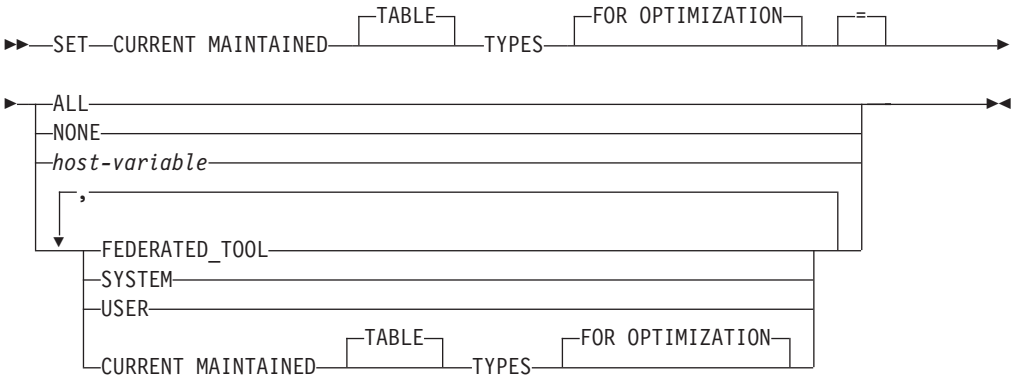
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

ALL
Specifies that all possible types of maintained tables controlled by this special register, now and in the future, are to be considered when optimizing the processing of dynamic SQL queries.

NONE
Specifies that none of the object types that are controlled by this special register are to be considered when optimizing the processing of dynamic SQL queries.

FEDERATED_TOOL
Specifies that refresh-deferred materialized query tables that are maintained by a federated tool can be considered to optimize the processing of dynamic SQL queries, provided the value of the CURRENT QUERY OPTIMIZATION special register is 2 or greater than 5.

SYSTEM
Specifies that system-maintained refresh-deferred materialized query tables can be considered to optimize the processing of dynamic SQL queries. (Immediate materialized query tables are always available.)

USER
Specifies that user-maintained refresh-deferred materialized query tables can be considered to optimize the processing of dynamic SQL queries.

CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
The value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register before this statement executes.

host-variable
A variable of type CHAR or VARCHAR. The length of the contents of the host variable must not exceed 254 bytes (SQLSTATE 42815). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of *host-variable* must be left-aligned. The contents of *host-variable* must be a string that is a comma-separated list of keywords matching what can be specified as keywords for the special register. These keywords must be specified in the exact case intended, because there is no conversion to uppercase characters. The value must be padded on the right with blanks if its length is less than that of the host variable.

Notes

- The initial value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register is SYSTEM.
- The CURRENT REFRESH AGE special register must be set to a value other than zero for the specified table types to be considered when optimizing the processing of dynamic SQL queries.

Examples

- *Example 1:* Set the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.

```
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION = SYSTEM, USER
```

- *Example 2:* Retrieve the current value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register into a host variable called CURMAINTYPES.

```
EXEC SQL VALUES (CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION)
  INTO :CURMAINTYPES
```

- *Example 3:* Set the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register to have no value.

```
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION = NONE
```

SET CURRENT MDC ROLLOUT MODE

The SET CURRENT MDC ROLLOUT MODE statement assigns a value to the CURRENT MDC ROLLOUT MODE special register. The value specifies the type of rollout cleanup that is to be performed on qualifying DELETE statements for multidimensional clustering (MDC) tables.

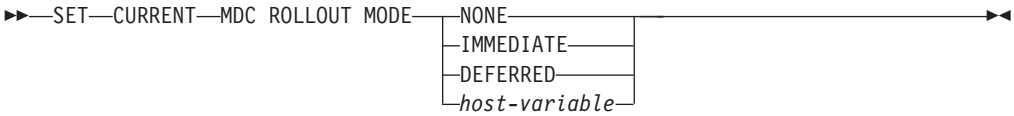
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

NONE
Specifies that MDC rollout optimization during delete operations is not to be used. The DELETE statement is processed in the same way as a DELETE statement that does not qualify for rollout.

IMMEDIATE
Specifies that MDC rollout optimization is to be used if the DELETE statement qualifies. If the table has RID indexes, the indexes are updated immediately during delete processing. The deleted blocks are available for reuse after the transaction commits.

DEFERRED
Specifies that MDC rollout optimization is to be used if the DELETE statement qualifies. If the table has RID indexes, index updates are deferred until after the transactions commits. With this option, delete processing is faster and uses less log space, but the deleted blocks are not available for reuse until after the index updates are complete.

host-variable
A variable of type VARCHAR. The length of *host-variable* must be less than or equal to 17 bytes (SQLSTATE 42815). The value of the host variable must be a left-aligned string that is one of 'NONE', 'IMMEDIATE', or 'DEFERRED' (case insensitive). If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

Notes

- Subsequent DELETE statements that are eligible for rollout processing respect the setting of the CURRENT MDC ROLLOUT MODE special register. Currently executing sections are not affected by a change to this special register.

- The effects of executing the SET CURRENT MDC ROLLOUT MODE statement are not rolled back if the unit of work in which the statement is executed is rolled back.
- In DB2 Version 9.7 and later releases, the DEFERRED mode is not supported on a data partitioned MDC table with partitioned RID indexes. Only the NONE and IMMEDIATE modes are supported. The cleanup rollout type will be IMMEDIATE if the **DB2_MDC_ROLLOUT** registry variable is set to DEFER, or if the CURRENT MDC ROLLOUT MODE special register is set to DEFERRED to override the **DB2_MDC_ROLLOUT** setting.

  If only nonpartitioned RID indexes exist on the MDC table, deferred index cleanup rollout is supported.

Example

Specify deferred cleanup behavior for the next DELETE statement that qualifies for rollout processing.

```
SET CURRENT MDC ROLLOUT MODE IMMEDIATE
```

SET CURRENT OPTIMIZATION PROFILE

The SET CURRENT OPTIMIZATION PROFILE statement assigns a value to the CURRENT OPTIMIZATION PROFILE special register. The value specifies the optimization profile the optimizer should use when preparing dynamic DML statements.

This statement is not under transaction control.

When the statement is evaluated, the name of the optimization profile is checked for validity, but the profile is not processed until the optimizer encounters a dynamic DML statement.

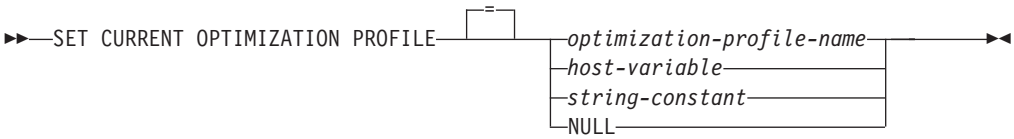
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

optimization-profile-name
The two-part name of the optimization profile. The name can be specified with a literal, host variable, or special register. The name specified is the name entered into the CURRENT OPTIMIZATION PROFILE special register.

If the specified optimization-profile-name is unqualified, the value of the CURRENT DEFAULT SCHEMA register is used as the implicit qualifier. The default value of the special register is null.

host-variable
A variable of type CHAR or VARCHAR that includes the name of the optimization profile. A host variable that includes a null indicator indicates that the value of the OPTPROFILE bind option is to be used if that value is specified for the current package. A host variable of zero length, or of white space only, indicates that no optimization profile is to be used.

The host variable must meet the following characteristics:

- The content of the string is a single or two-part identifier (separated by a period), with no leading blanks.
- The identifier or identifiers can be delimited or non-delimited.
- The content of the string is not folded to upper case.
- Lower case and special characters cannot be used in non-delimited strings.

- If the first character is a double quotation mark, a closing double quotation mark must either precede a period or be the last non-blank character in the string.
- If the first character following a period is a double quotation mark, then a double quotation mark must be the last non-blank character in the string.
- If the identifier is delimited, then to include double quotation marks in the identifier, specify the character twice.
- Any period that is not inside a delimited identifier is treated as a separator, and only one period separator can exist in the string.

string-constant
Specifies a constant as a character string that is the name of the optimization profile. The content of a string constant must meet the same characteristics as a host variable.

NULL
Sets the CURRENT OPTIMIZATION PROFILE register to null.

Table 35 provides examples of string literals and identifiers that might be used to assign the register as per the optimization profile naming rules. The value in the SCHEMA and NAME column represent an optimization profile name as it might appear in the OPT_PROFILE table. The valid string literals column shows string literals that match the optimization profile named by the corresponding SCHEMA and NAME column values. The valid identifiers column shows identifiers that would identify that same optimization profile.

Table 35. Examples of string literals and identifiers

SCHEMA	NAME	Valid string literals	Valid identifiers
SIMMEN	BIG_PROF	'BIG_PROF'	BIG_PROF
		'SIMMEN.BIG_PROF'	SIMMEN.BIG_PROF
		'"BIG_PROF"'	"BIG_PROF"
		'"SIMMEN"."BIG_PROF"'	"SIMMEN"."BIG_PROF"
SIMMEN	low_profile	'"low_profile"'	"low_profile"
		'SIMMEN."low_profile"'	SIMMEN."low_profile"
		'"SIMMEN"."low_profile"'	"SIMMEN"."low_profile"
eliaz	DBA3	'DBA3'	DBA3
		'"DBA3"'	"eliaz".DBA3
		'"eliaz".DBA3'	"eliaz"."DBA3"
		'"eliaz"."DBA3"'	
SNOW	PROFILE1.0	'"PROFILE1.0"'	"PROFILE1.0"
		'SNOW."PROFILE1.0"'	SNOW."PROFILE1.0"
		'"SNOW"."PROFILE1.0"'	"SNOW"."PROFILE1.0"

Notes

- If the value of the register specifies the name of an existing optimization profile, the specified optimization profile is used when preparing subsequent dynamic DML statements.

- If the value of the register is null, the optimization profile specified by the OPTPROFILE bind option, if any, is used when preparing subsequent dynamic DML statements.
- If the value of the register is null, and the OPTPROFILE bind option is not set, no optimization profile is used when preparing subsequent dynamic DML statements.
- If the value of the register is the empty string, then no optimization profile is used when preparing subsequent dynamic DML statements, regardless of whether the OPTPROFILE bind option is set.
- Subsequent changes to CURRENT DEFAULT SCHEMA do not have any effect on the optimization profile. The CURRENT OPTIMIZATION PROFILE register value is set with the two part name that is in effect at the time SET CURRENT OPTIMIZATION PROFILE statement is evaluated. Only another SET CURRENT OPTIMIZATION PROFILE statement can change the optimization profile that is used.

Examples

- *Example 1:* The optimization profile RICK.FOO is used for statements 1, 2, and 3. TOM.FOO is used for statement 4.

```
SET CURRENT SCHEMA = 'RICK'
SET CURRENT OPTIMIZATION PROFILE = 'FOO'
   statement 1
   statement 2
SET CURRENT SCHEMA = 'TOM'
   statement 3
SET CURRENT OPTIMIZATION PROFILE = 'FOO'
   statement 4
```

- *Example 2:* An application with the following statements was bound with the options OPTPROFILE("Foo") and QUALIFIER("John"). The optimization profile KAAREL.BAR is used for statement 1 and optimization profile "John"."Foo" is used for statement 2.

```
SET CURRENT SCHEMA = 'KAAREL'
SET CURRENT OPTIMIZATION PROFILE = 'BAR'
   statement 1
SET CURRENT SCHEMA = "Tom"
SET CURRENT OPTIMIZATION PROFILE NULL
   statement 2
```

- *Example 3:* The empty string is a special value that indicates that no optimization profile is to be used. Optimization profile "Hamid"."Foo" is used for statement 1 and no optimization profile is used for statement 2.

```
SET CURRENT OPTIMIZATION PROFILE = '"Hamid"."Foo"'
   statement 1
SET CURRENT OPTIMIZATION PROFILE = ''
   statement 2
```

SET CURRENT PACKAGE PATH

The SET CURRENT PACKAGE PATH statement assigns a value to the CURRENT PACKAGE PATH special register.

This statement is not under transaction control.

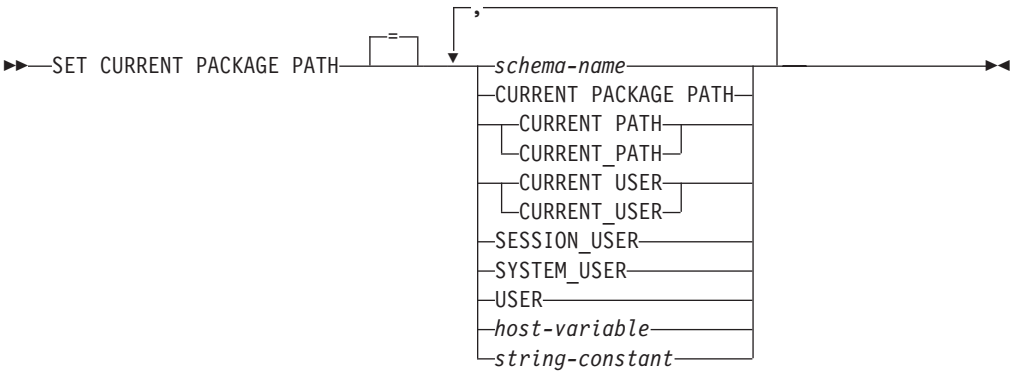
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax



Description

schema-name
Identifies a schema. The name must not be a delimited identifier that is empty or that contains only blanks (SQLSTATE 42815).

CURRENT PACKAGE PATH
The value of the CURRENT PACKAGE PATH special register before this statement executes.

CURRENT PATH
The value of the CURRENT PATH special register.

CURRENT USER
The value of the CURRENT USER special register.

SESSION_USER
The value of the SESSION_USER special register.

SYSTEM_USER
The value of the SYSTEM_USER special register.

USER
The value of the USER special register.

host-variable
Contains one or more schema names, separated by commas. The host variable must:

- Be a character-string variable (CHAR or VARCHAR). The actual length of the contents of the host variable must not exceed the length of the CURRENT PACKAGE PATH special register.
- Not be the null value. If an indicator variable is provided, its value must not indicate a null value.
- Contain an empty or blank string, or one or more schema names separated by commas.
- Be padded on the right with blanks if the actual length of the host variable is greater than the content.
- Not contain CURRENT PACKAGE PATH, CURRENT PATH, CURRENT_PATH, CURRENT USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, PATH, or USER.
- Not contain a delimited identifier that is empty or that contains only blanks.

string-constant
Specifies a character string constant that contains zero, one, or more schema names that are separated by commas. The string constant must:

- Have a length that does not exceed the maximum length of the CURRENT PACKAGE PATH special register.
- Not contain CURRENT PACKAGE PATH, CURRENT PATH, CURRENT_PATH, CURRENT USER, CURRENT_USER, SESSION_USER, SYSTEM_USER, PATH, or USER.
- Not contain a delimited identifier that is empty or that contains only blanks.

Rules

- If the same schema appears more than once in the list, the first occurrence of the schema is used (SQLSTATE 01625).
- The number of schemas that can be specified is limited by the total length of the CURRENT PACKAGE PATH special register. The special register string is built by taking each specified schema name and removing trailing blanks, delimiting the name with double quotation marks, and separating the schema names with commas. The length of the resulting list cannot exceed the maximum length of the special register (SQLSTATE 0E000).
- A schema name that does not conform to the rules for an ordinary identifier (for example, a schema name that contains lowercase characters or characters that cannot be specified in an ordinary identifier), must be specified as a delimited schema name, and must not be specified within a host variable or string constant.
- To indicate that the current value of a special register (specified as a single keyword) is to be used in the package path, specify the name of the special register as a keyword. If the name of the special register is specified as a delimited identifier instead (for example, "USER"), it is interpreted as a schema name of that value ('USER').
- The following rules are used to determine whether a value specified in a SET CURRENT PACKAGE PATH statement is a variable or a schema name:
  - If *name* is the same as a parameter or SQL variable in the SQL procedure, *name* is interpreted as a parameter or SQL variable, and the value in *name* is assigned to the package path.
  - If *name* is not the same as a parameter or SQL variable in the SQL procedure, *name* is interpreted as a schema name, and the value in *name* is assigned to the package path.

Notes

- ***Transaction considerations:*** The SET CURRENT PACKAGE PATH statement is not a commitable operation. ROLLBACK has no effect on the CURRENT PACKAGE PATH special register.
- ***Existence checking of schemas:*** No validation that the specified schemas exist is made at the time that the CURRENT PACKAGE PATH special register is set. For example, a schema that is misspelled is not detected, which could affect the way subsequent SQL operates. At package execution time, authorization to a matching package is checked, and if this authorization check fails, an error is returned (SQLSTATE 42501).
- ***Contents of host variable or string constant:*** The contents of a host variable or a string constant are interpreted as a list of schema names. If multiple schema names are specified, they must be separated by commas. Each schema name in the list must conform to the rules for forming an ordinary identifier, or be specified as a delimited identifier. The contents of the host variable or string constant are not folded to uppercase.
- ***Restrictions specific to embedded SQL for COBOL applications:*** A maximum of ten literal (non-host variable) values can appear on the right side of a SET CURRENT PACKAGE PATH statement. Such values can have a maximum length of 130 (non-delimited) or 128 (delimited).

Examples

- *Example 1:* Set the CURRENT PACKAGE PATH special register to the following list of schemas: MYPKGS, 'ABC E', SYSIBM

  ```
  SET CURRENT PACKAGE PATH = MYPKGS, 'ABC E', SYSIBM
  ```

  The following statement sets a host variable to the value of the resulting list:

  ```
  SET :hvpklist = CURRENT PACKAGE PATH
  ```

  The value of the host variable is: "MYPKGS", "ABC E", "SYSIBM".
- *Example 2:* Set the CURRENT PACKAGE PATH special register to the following list of schemas: "SCH4","SCH5", where :hvar1 contains 'SCH4,SCH5'.

  ```
  SET CURRENT PACKAGE PATH :hvar1
  ```

  The value of the CURRENT PACKAGE PATH special register after this statement executes is: "SCH4","SCH5".
- *Example 3:* Set the CURRENT PACKAGE PATH special register to the following list of schemas: "SCH1","SCH#2","SCH3","SCH4","SCH5", where :hvar1 contains 'SCH4,SCH5'.

  ```
  SET CURRENT PACKAGE PATH = SCH1,'SCH#2',"SCH3",:hvar1
  ```

  The value of the CURRENT PACKAGE PATH special register after this statement executes is: "SCH1","SCH#2","SCH3","SCH4","SCH5".
- *Example 4:* Clear the CURRENT PACKAGE PATH special register.

  ```
  SET CURRENT PACKAGE PATH = ''
  ```
- *Example 5:* Temporarily append the "SCH_PROD" schema (contained in the :prodschema host variable) and the "SCH_PROD2" schema (contained in the :prod2schema host variable) to the end of the CURRENT PACKAGE PATH special register for execution of the SUMMARIZE procedure. Then, switch the CURRENT PACKAGE PATH special register back to its previous value.

  ```
  SET :oldCPP = CURRENT PACKAGE PATH

  SET CURRENT PACKAGE PATH = CURRENT PACKAGE PATH,:prodschema,:prod2schema
  ```

```
CALL SUMMARIZE(:V1,:V2)

SET CURRENT PACKAGE PATH = :oldCPP
```

- *Example 6:* Set the CURRENT PACKAGE PATH special register to a list of delimited schema names: "MY.SCHEMA" (imbedded period), "OLD SCHEMA" (imbedded blank). Use a single host variable containing both delimited identifiers:

  ```
  hv = '"MY.SCHEMA", "OLD SCHEMA"'

  SET CURRENT PACKAGE PATH = :hv
  ```

  or use a single string constant containing both delimited identifiers:

  ```
  SET CURRENT PACKAGE PATH = '"MY.SCHEMA", "OLD SCHEMA"'
  ```

  or use a list of delimited schemas:

  ```
  SET CURRENT PACKAGE PATH = 'MY.SCHEMA', 'OLD SCHEMA'
  ```

SET CURRENT PACKAGESET

The SET CURRENT PACKAGESET statement sets the schema name (collection identifier) that will be used to select the package to use for subsequent SQL statements.

This statement is not under transaction control.

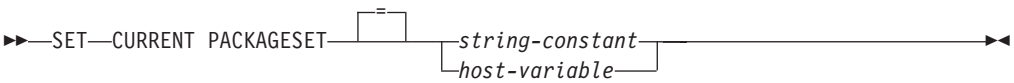
Invocation

This statement can be embedded only in an application program. It is an executable statement that cannot be dynamically prepared. This statement is not supported in REXX.

Authorization

None required.

Syntax



Description

string-constant
A character string constant. If the value exceeds 128 bytes, only the first 128 bytes are used.

host-variable
A variable of type CHAR or VARCHAR. It cannot be set to null. If the value exceeds 128 bytes, only the first 128 bytes are used.

Notes

- This statement allows an application to specify the schema name used when selecting a package for an executable SQL statement. The statement is processed at the client and does not flow to the application server.
- The COLLECTION bind option can be used to create a package with a specified schema name.
- Unlike DB2 for z/OS, the SET CURRENT PACKAGESET statement is implemented without support for a special register called CURRENT PACKAGESET.

Examples

- *Example 1*: Assume an application called TRYIT is precompiled by user ID PRODUSA, making 'PRODUSA' the default schema name in the bind file. The application is then bound twice with different bind options. The following command line processor commands were used:

```
DB2 CONNECT TO SAMPLE USER PRODUSA
DB2 BIND TRYIT.BND DATETIME USA
DB2 CONNECT TO SAMPLE USER PRODEUR
DB2 BIND TRYIT.BND DATETIME EUR COLLECTION 'PRODEUR'
```

This creates two packages called TRYIT. The first bind command created the package in the schema named 'PRODUSA'. The second bind command created the package in the schema named 'PRODEUR' based on the COLLECTION option.

- *Example 2*: Assume the application TRYIT contains the following statements:

```
EXEC SQL CONNECT TO SAMPLE;
.
.
EXEC SQL SELECT HIREDATE INTO :HD FROM EMPLOYEE WHERE EMPNO='000010'; 1
.
.
EXEC SQL SET CURRENT PACKAGESET 'PRODEUR';                            2
.
.
EXEC SQL SELECT HIREDATE INTO :HD FROM EMPLOYEE WHERE EMPNO='000010'; 3
```

1 This statement will run using the PRODUSA.TRYIT package because it is the default package for the application. The date is therefore returned in USA format.

2 This statement sets the schema name to 'PRODEUR' for package selection.

3 This statement will run using the PRODEUR.TRYIT package as a result of the SET CURRENT PACKAGESET statement. The date is therefore returned in EUR format.

SET CURRENT QUERY OPTIMIZATION

The SET CURRENT QUERY OPTIMIZATION statement assigns a value to the CURRENT QUERY OPTIMIZATION special register. The value specifies the current class of optimization techniques enabled when preparing dynamic SQL statements.

This statement is not under transaction control.

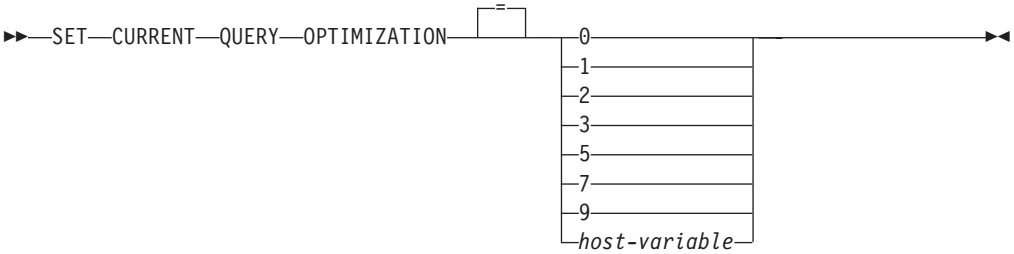
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

optimization-class

optimization-class can be specified either as an integer constant or as the name of a host variable that will contain the appropriate value at run time. An overview of the classes follows.

0 Specifies that a minimal amount of optimization is performed to generate an access plan. This class is most suitable for simple dynamic SQL access to well-indexed tables.

1 Specifies that optimization roughly comparable to DB2 Version 1 is performed to generate an access plan.

2 Specifies a level of optimization higher than that of DB2 Version 1, but at significantly less optimization cost than levels 3 and higher, especially for very complex queries.

3 Specifies that a moderate amount of optimization is performed to generate an access plan.

5 Specifies a significant amount of optimization is performed to generate an access plan. For complex dynamic SQL queries, heuristic rules are used to limit the amount of time spent selecting an access plan. Where possible, queries will use materialized query tables instead of the underlying base tables.

7 Specifies a significant amount of optimization is performed to generate an access plan. Similar to 5 but without the heuristic rules.

9 Specifies a maximal amount of optimization is performed to generate an access plan. This can greatly expand the number of possible access plans that are evaluated. This class should be used to determine if a better access plan can be generated for very complex and very long-running queries using large tables. Explain and performance measurements can be used to verify that a better plan has been generated.

host-variable
The data type is INTEGER. The value must be in the range 0 to 9 (SQLSTATE 42815) but should be 0, 1, 2, 3, 5, 7, or 9. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

Notes

- When the CURRENT QUERY OPTIMIZATION register is set to a particular value, a set of query rewrite rules are enabled, and certain optimization variables take on particular values. This class of optimization techniques is then used during preparation of dynamic SQL statements.
- In general, changing the optimization class impacts the execution time of the application, the compilation time, and resources required. Most statements will be adequately optimized using the default query optimization class. Lower query optimization classes, especially classes 1 and 2, may be appropriate for dynamic SQL statements for which the resources consumed by the dynamic *PREPARE* are a significant portion of those required to execute the query. Higher optimization classes should be chosen only after considering the additional resources that may be consumed and verifying that a better access plan has been generated.
- Query optimization classes must be in the range 0 to 9. Classes outside this range will return an error (SQLSTATE 42815). Unsupported classes within this range will return a warning (SQLSTATE 01608) and will be replaced with the next lowest query optimization class. For example, a query optimization class of 6 will be replaced by 5.
- Dynamically prepared statements use the class of optimization that was set by the most recently executed SET CURRENT QUERY OPTIMIZATION statement. In cases where a SET CURRENT QUERY OPTIMIZATION statement has not yet been executed, the query optimization class is determined by the value of the **dft_queryopt** database configuration parameter.
- Statically bound statements do not use the CURRENT QUERY OPTIMIZATION special register; therefore this statement has no effect on them. The QUERYOPT option is used during preprocessing or binding to specify the required class of optimization for statically bound statements. If QUERYOPT is not specified then, the default value specified by the **dft_queryopt** database configuration parameter is used.
- The results of executing the SET CURRENT QUERY OPTIMIZATION statement are not rolled back if the unit of work in which it is executed is rolled back.

Examples

- *Example 1:* This example shows how the highest degree of optimization can be selected.

```
SET CURRENT QUERY OPTIMIZATION 9
```

- *Example 2:* The following example shows how the CURRENT QUERY OPTIMIZATION special register can be used within a query.

  Using the SYSCAT.PACKAGES catalog view, find all plans that were bound with the same setting as the current value of the CURRENT QUERY OPTIMIZATION special register.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT PKGNAME, PKGSCHEMA FROM SYSCAT.PACKAGES
  WHERE QUERYOPT = CURRENT QUERY OPTIMIZATION
```

SET CURRENT REFRESH AGE

The SET CURRENT REFRESH AGE statement changes the value of the CURRENT REFRESH AGE special register.

This statement is not under transaction control.

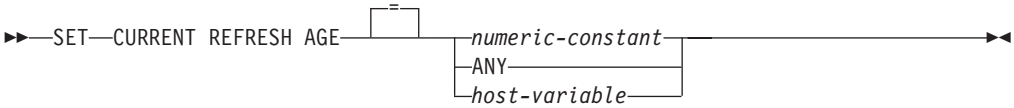
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

numeric-constant
A DECIMAL(20,6) value representing a timestamp duration. The value must be 0 or 99 999 999 999 999 (the microseconds portion of the value is ignored and can therefore be any value).

ANY
This is a shorthand for 99 999 999 999 999.

host-variable
A variable of type DECIMAL(20,6) or another type that is assignable to DECIMAL(20,6). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815). The value of *host-variable* must be 0 or 99 999 999 999 999.

Notes

- The initial value of the CURRENT REFRESH AGE special register is zero.
- The value of CURRENT REFRESH AGE is replaced by the specified value. The value must be 0 or 99 999 999 999 999. The value 99 999 999 999 999 represents 9999 years, 99 months, 99 days, 99 hours, 99 minutes, and 99 seconds.

  If the value of CURRENT REFRESH AGE is 0, the materialized query tables affected by this special register will not be used to optimize the processing of a query. If the value of CURRENT REFRESH AGE is 99 999 999 999 999, the materialized query tables affected by this special register can be used to optimize the processing of a query, but only if the value of CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register includes them, and the CURRENT QUERY OPTIMIZATION special register is set to 2 or a value greater than or equal to 5. The materialized query tables affected by this special register are REFRESH DEFERRED MAINTAINED BY USER and REFRESH DEFERRED MAINTAINED BY SYSTEM.

REFRESH IMMEDIATE MAINTAINED BY SYSTEM materialized query tables can always be used to optimize the processing of a query if the CURRENT QUERY OPTIMIZATION special register is set to 2 or a value greater than or equal to 5.

REFRESH DEFERRED MAINTAINED BY FEDERATED_TOOL materialized query tables are used for optimization if the CURRENT QUERY OPTIMIZATION special register is set to 2 or a value greater than or equal to 5, and the value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register is set to ALL or includes FEDERATED_TOOL.

- Setting the CURRENT REFRESH AGE special register to a value other than zero should be done with caution. A table type specified by the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register might not represent the values of the underlying base table. If such a table is used to optimize the processing of a query, the query result might *not* accurately represent the data in the underlying table. This might be reasonable if you know that the underlying data has not changed, or if you are willing to accept a degree of error in the results, based on your knowledge of the cached data.
- The CURRENT REFRESH AGE value of 99 999 999 999 999 cannot be used in timestamp arithmetic operations, because the result would be outside of the valid range for dates (SQLSTATE 22008).

Examples

- *Example 1:* The following statement sets the CURRENT REFRESH AGE special register.

```
SET CURRENT REFRESH AGE ANY
```

- *Example 2:* The following example retrieves the value of the CURRENT REFRESH AGE special register into a host variable called CURMAXAGE. The value, set by the previous example, is 99999999999999.000000.

```
EXEC SQL VALUES (CURRENT REFRESH AGE) INTO :CURMAXAGE;
```

SET CURRENT SQL_CCFLAGS

The SET CURRENT SQL_CCFLAGS statement changes the value of the CURRENT SQL_CCFLAGS special register.

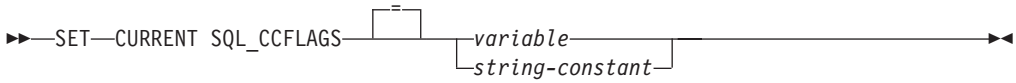
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

variable
Specifies a variable that contains one or more name and value pairs that are separated by commas.

The variable must have the following characteristics (SQLSTATE 42815):

- The data type must be CHAR or VARCHAR. The actual length of the contents of the variable must not exceed the maximum length of the special register.
- It must be a string of blanks, an empty string, or include one or more name and value pairs where the name is separated from the value by the colon character. The name must be a valid ordinary identifier. The value associated with a name must be a BOOLEAN constant, an INTEGER constant, or the keyword NULL.
- It must be padded on the right with blanks if using a fixed-length character variable.
- It can include extra blanks at the beginning or ending of the string, around the comma character, or around the colon character. The blanks are ignored.
- It must not be the null value.

string-constant
Specifies a character string constant that contains one or more name and value pairs that are separated by commas.

The string constant must have the following characteristics (SQLSTATE 42815):

- It must be a character string constant. The length of the constant must not exceed the maximum length of the special register.
- It must be a string of blanks, an empty string or include one or more name and value pairs where the name is separated from the value by the colon character. The name must be a valid ordinary identifier. The value associated with a name must be a BOOLEAN constant, an INTEGER constant, or the keyword NULL.
- It can include extra blanks at the beginning or ending of the string, around the comma character, or around the colon character. The blanks are ignored.

Notes

- If a duplicate name appears in the content for the CURRENT SQL_FLAGS special register, then only the last (furthest to the right) value is used. The special register value will include only a single occurrence of the duplicated name with the value that is used. Concatenating a duplicated name with a different value to the CURRENT SQL_CCFLAGS value can be used to override some conditional compilation values while retaining other values.
- When the CURRENT SQL_CCFLAGS is retrieved, the returned string includes the unique name and value pairs in uppercase characters with multiple pairs separated by a comma and a blank. The pairs are in the order they were specified, with a duplicate name appearing only where it first occurred, but reflecting the value from where it last occurred.
- The CURRENT SQL_CCFLAGS special register can be set to the default defined for the database by retrieving the VALUE column from SYSIBMADM.DBCFG where NAME='sql_ccflags' into a variable and then assigning that variable to the special register.
- ***Transaction considerations:*** The SET SQL_CCFLAGS statement is not a committable operation. ROLLBACK has no effect on CURRENT SQL_CCFLAGS.

Examples

- *Example 1*: Define a conditional compilation value for the session to indicate that the server is DB2 9.7 and that debug is false.

```
SET CURRENT SQL_CCFLAGS 'db2v97:true, debug:false'
```

- *Example 2*: Extend the existing CURRENT SQL_CCFLAGS to set debug to true and define the tracing level.

```
BEGIN
  DECLARE LIST VARCHAR(1024);
  SET LIST = CASE WHEN (CURRENT SQL_CCFLAGS = ' ')
              THEN 'tracelvl:3,debug:true'
              ELSE CURRENT SQL_CCFLAGS
                   concat ',tracelvl:3,debug:true'
              END;
  SET CURRENT SQL_CCFLAGS = LIST;
END
```

A CASE expression is used in the assignment to handle the possibility that the CURRENT SQL_CCFLAGS special register does not include any conditional compilation values, resulting in a leading comma in the value of the variable LIST.

A query of the CURRENT SQL_CCFLAGS special register after the execution of the statement in Example 1 and the compound statement in this example would return:

```
DB2V97:TRUE, DEBUG:TRUE, TRACELVL:3
```

Even though the conditional compilation value for DEBUG appeared twice in the variable LIST, it appears only once in the special register value where it would have first appeared.

SET CURRENT TEMPORAL BUSINESS_TIME

The SET CURRENT TEMPORAL BUSINESS_TIME statement changes the value of the CURRENT TEMPORAL BUSINESS_TIME special register.

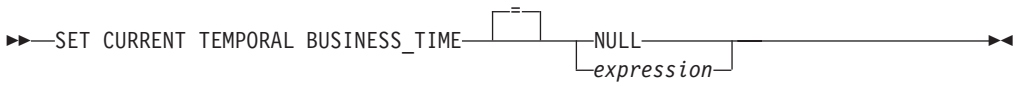
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

NULL
Specifies the null value.

expression
Each expression can contain any of the following supported operands (SQLSTATE 428HY):

- Constant
- Special register
- Variable (host variable, SQL variable, SQL parameter, transition variable, global variable)
- Built-in scalar function whose arguments are supported operands. User-defined functions and non-deterministic functions are not supported in this context.
- CAST specification where the cast operand is a supported operand
- Expression using arithmetic operator and operands

Notes

- ***Transaction considerations:*** The SET CURRENT TEMPORAL BUSINESS_TIME statement is not a committable operation. ROLLBACK has no effect on CURRENT TEMPORAL BUSINESS_TIME.
- ***Effects on other special registers:*** The setting of the CURRENT TEMPORAL BUSINESS_TIME special register does not have any effect on the values of other special registers, specifically the CURRENT DATE and CURRENT TIMESTAMP special registers.

Examples

- *Example 1:* Set the CURRENT TEMPORAL BUSINESS_TIME special register to the previous month.

```
SET CURRENT TEMPORAL BUSINESS_TIME = CURRENT TIMESTAMP - 1 MONTH
```

- *Example 2:* Set the CURRENT TEMPORAL BUSINESS_TIME special register to the null value.

```
SET CURRENT TEMPORAL BUSINESS_TIME = NULL
```

SET CURRENT TEMPORAL SYSTEM_TIME

The SET CURRENT TEMPORAL SYSTEM_TIME statement changes the value of the CURRENT TEMPORAL SYSTEM_TIME special register.

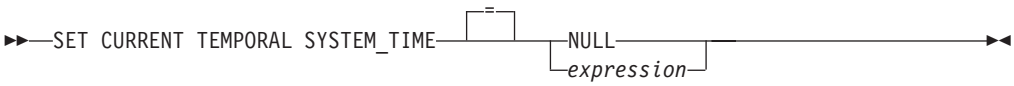
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

NULL
Specifies the null value.

expression
Each expression can contain any of the following supported operands (SQLSTATE 428HY):

- Constant
- Special register
- Variable (host variable, SQL variable, SQL parameter, transition variable, global variable)
- Built-in scalar function whose arguments are supported operands. User-defined functions and non-deterministic functions are not supported in this context.
- CAST specification where the cast operand is a supported operand
- Expression using arithmetic operator and operands

Notes

- ***Transaction considerations:*** The SET CURRENT TEMPORAL SYSTEM_TIME statement is not a committable operation. ROLLBACK has no effect on CURRENT TEMPORAL SYSTEM_TIME.
- ***Effects on other special registers:*** The setting of the CURRENT TEMPORAL SYSTEM_TIME special register does not have any effect on the values of other special registers, specifically the CURRENT DATE and CURRENT TIMESTAMP special registers.

Examples

- *Example 1:* Set the CURRENT TEMPORAL SYSTEM_TIME special register to the previous month.

```
SET CURRENT TEMPORAL SYSTEM_TIME = CURRENT TIMESTAMP - 1 MONTH
```

- *Example 2:* Set the CURRENT TEMPORAL SYSTEM_TIME special register to the null value.

```
SET CURRENT TEMPORAL SYSTEM_TIME = NULL
```

SET ENCRYPTION PASSWORD

The SET ENCRYPTION PASSWORD statement sets the password to be used by the ENCRYPT, DECRYPT_BIN and DECRYPT_CHAR functions. The password is not tied to DB2 authentication, and is used for data encryption and decryption only.

This statement is not under transaction control.

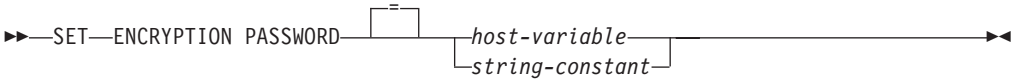
Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

The encryption password can be used by the ENCRYPT, DECRYPT_BIN, and DECRYPT_CHAR built-in functions for password-based encryption. The length of the password must be between 6 and 127 bytes and all characters must be specified in the exact case intended, because there is no automatic conversion to uppercase characters. To maintain the best level of security on your system, it is recommended that you use a host variable or dynamic parameter markers to specify the password, rather than using a literal string in your SET ENCRYPTION PASSWORD statement.

host-variable
A variable of type CHAR or VARCHAR. The length of the *host-variable* must be between 6 and 127 bytes (SQLSTATE 428FC). It cannot be set to null. All characters are specified in the exact case intended, as there is no conversion to uppercase characters.

string-constant
A character string constant. The length must be between 6 and 127 bytes (SQLSTATE 428FC).

Notes

- The initial value of the ENCRYPTION PASSWORD is the empty string.
- The *host-variable* or *string-constant* is transmitted to the database server using normal DB2 mechanisms.

Example

The following example shows how you can set the ENCRYPTION PASSWORD special register in an embedded SQL application using parameter markers. It is strongly recommended that this special register is always set up using parameter markers in your applications.

```
EXEC SQL BEGIN DECLARE SECTION;
      char hostVarSetEncPassStmt[200];
      char hostVarPassword[128];
EXEC SQL END DECLARE SECTION;

/* prepare the statement with a parameter marker */
strcpy(hostVarSetEncPassStmt, "SET ENCRYPTION PASSWORD = ?");
EXEC SQL PREPARE hostVarSetEncPassStmt FROM :hostVarSetEncPassStmt;

/* execute the statement for hostVarPassword = 'Gre89Ea' */
strcpy(hostVarPassword, "Gre89Ea");
EXEC SQL EXECUTE hostVarSetEncPassStmt USING :hostVarPassword;
```

SET EVENT MONITOR STATE

The SET EVENT MONITOR STATE statement activates or deactivates an event monitor. The current state of an event monitor (active or inactive) is determined by using the EVENT_MON_STATE built-in function.

The SET EVENT MONITOR STATE statement is not under transaction control.

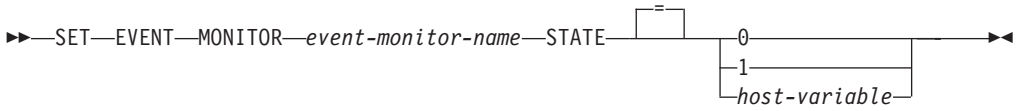
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include DBADM or SQLADM authority.

Syntax



Description

event-monitor-name
Identifies the event monitor to activate or deactivate. The name must identify an event monitor that exists in the catalog (SQLSTATE 42704).

new-state
new-state can be specified either as an integer constant or as the name of a host variable that will contain the appropriate value at run time. The following values can be specified:

0 Indicates that the specified event monitor should be deactivated.

1 Indicates that the specified event monitor should be activated. The event monitor should not already be active; otherwise a warning (SQLSTATE 01598) is issued.

host-variable
The data type is INTEGER. The value specified must be 0 or 1 (SQLSTATE 42815). If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

Rules

- Although an unlimited number of event monitors may be defined, a maximum of 128 event monitors can be active simultaneously on each database partition. In a multiple partition database environment, a maximum of 32 GLOBAL event monitors can be active simultaneously on each database.

- In order to activate an event monitor, the transaction in which the event monitor was created must have been committed (SQLSTATE 55033). This rule prevents (in one unit of work) creating an event monitor, activating the monitor, then rolling back the transaction.
- If the number or size of the event monitor files exceeds the values specified for MAXFILES or MAXFILESIZE on the CREATE EVENT MONITOR statement, an error (SQLSTATE 54031) is raised.
- If the target path of the event monitor (that was specified on the CREATE EVENT MONITOR statement) is already in use by another event monitor, an error (SQLSTATE 51026) is raised.

Notes

- Activating a non-WLM event monitor performs a reset of any counters associated with it. The reset of counters does not occur when activating WLM, locking, and unit of work event monitors.
- When a WRITE TO TABLE event monitor is started using SET EVENT MONITOR STATE, it updates the EVMON_ACTIVATES column of the SYSCAT.EVENTMONITORS catalog view. If the unit of work in which the set operation was performed is rolled back for any reason, that catalog update is lost. When the event monitor is restarted, it will reuse the EVMON_ACTIVATES value that was rolled back.
- If the database partition on which the event monitor is to run is not active, event monitor activation occurs when that database partition next activates.
- After an event monitor is activated, it behaves like an autostart event monitor until that event monitor is explicitly deactivated or the instance is recycled. That is, if an event monitor is active when a database partition is deactivated, and that database partition is subsequently reactivated, the event monitor is also explicitly reactivated.
- If an activity event monitor is active when the database deactivates, any backlogged activity records in the queue are discarded. To ensure that you obtain all activity event monitor records and that none are discarded, explicitly deactivate the activity event monitor first before deactivating the database. When an activity event monitor is explicitly deactivated, all backlogged activity records in the queue are processed before the event monitor deactivates.

Examples

- *Example 1:* Activate an event monitor named SMITHPAY.

```
SET EVENT MONITOR SMITHPAY STATE = 1
```

- *Example 2:* Assume that MYSAMPLE is a multiple partition database with two database partitions, 0 and 2. Partition 2 is not yet active.

```
On database partition 0:

CONNECT TO MYSAMPLE;
CREATE EVENT MONITOR MYEVMON ON DBPARTITIONNUM 2;
SET EVENT MONITOR MYEVMON STATE 1;
```

MYEVMON automatically activates whenever MYSAMPLE activates on database partition 2. This occurs until **SET EVENT MONITOR** MYEVMON **STATE** 0 is issued, or partition 2 is stopped.

SET INTEGRITY

The SET INTEGRITY statement is used to set the integrity pending state on tables, place tables into full access state, and prune the contents of one or more staging tables.

The following operations can be performed with the SET INTEGRITY statement:

- Bring one or more tables out of set integrity pending state (previously known as "check pending state") by performing required integrity processing on those tables.
- Bring one or more tables out of set integrity pending state without performing required integrity processing on those tables.
- Place one or more tables in set integrity pending state.
- Place one or more tables into full access state.
- Prune the contents of one or more staging tables.

When the statement is used to perform integrity processing for a table after it has been loaded or attached, the system can incrementally process the table by checking only the appended portion for constraints violations. If the subject table is a materialized query table or a staging table, and load, attach, or detach operations are performed on its underlying tables, the system can incrementally refresh the materialized query table or incrementally propagate to the staging table with only the delta portions of its underlying tables. However, there are some situations in which the system will not be able to perform such optimizations and will instead perform full integrity processing to ensure data integrity. Full integrity processing is done by checking the entire table for constraints violations, recomputing a materialized query table's definition, or marking a staging table as inconsistent. The latter implies that a full refresh of its associated materialized query table is required. There is also a situation in which you might want to explicitly request incremental processing by specifying the INCREMENTAL option.

The SET INTEGRITY statement is under transaction control.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges required to execute the SET INTEGRITY statement depend on the purpose, as outlined in the following list.

- Bringing tables out of set integrity pending state and performing the required integrity processing.

  The privileges held by the authorization ID of the statement must include at least one of the following:

  – CONTROL privilege on:

    - The tables on which integrity processing is performed and, if exception tables are provided for one or more of those tables, INSERT privilege on the exception tables

    - All descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables that will implicitly be placed in set integrity pending state by the statement
  - LOAD authority (with conditions). The following conditions must all be met before LOAD authority can be considered as providing valid privileges:
    - The required integrity processing does not involve the following actions:
      - Refreshing a materialized query table
      - Propagating to a staging table
      - Updating a generated or identity column
    - If exception tables are provided for one or more tables, the required access is granted for the duration of the integrity processing to the tables on which integrity processing is performed, and to the associated exception tables. That is:
      - SELECT and DELETE privilege on each table on which integrity processing is performed, and
      - INSERT privilege on the exception tables
  - DATAACCESS authority
- Bringing tables out of set integrity pending state without performing the required integrity processing.

  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on the tables that are being processed; CONTROL privilege on each descendent foreign key table, descendent immediate materialized query table, and descendent immediate staging table that will implicitly be placed in set integrity pending state by the statement
  - LOAD authority
  - DATAACCESS authority
  - DBADM authority
- Placing tables in set integrity pending state.

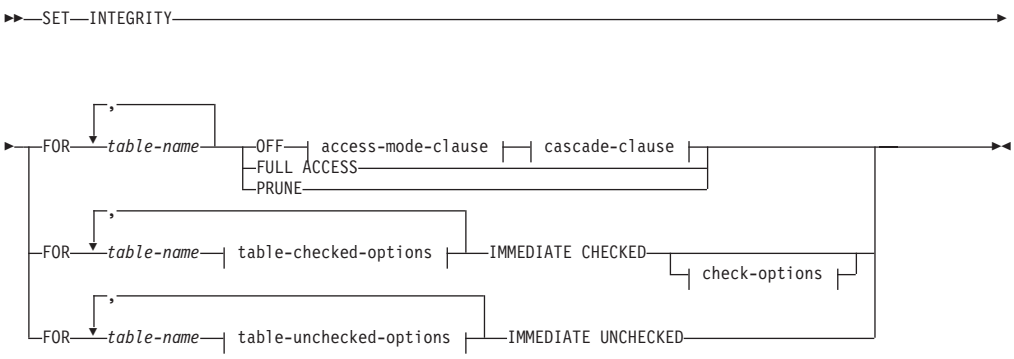
  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on:
    - The specified tables, and
    - The descendent foreign key tables that will be placed in set integrity pending state by the statement, and
    - The descendent immediate materialized query tables that will be placed in set integrity pending state by the statement, and
    - The descendent immediate staging tables that will be placed in set integrity pending state by the statement
  - LOAD authority
  - DATAACCESS authority
  - DBADM authority
- Place a table into the full access state.

  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on the tables that are placed into the full access state
  - LOAD authority
  - DATAACCESS authority

  - DBADM authority
- Prune a staging table.

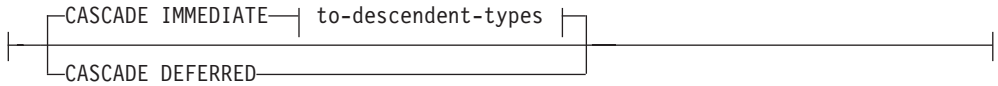
  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on the table being pruned
  - DATAACCESS authority

Syntax



access-mode-clause:

cascade-clause:



to-descendent-types:

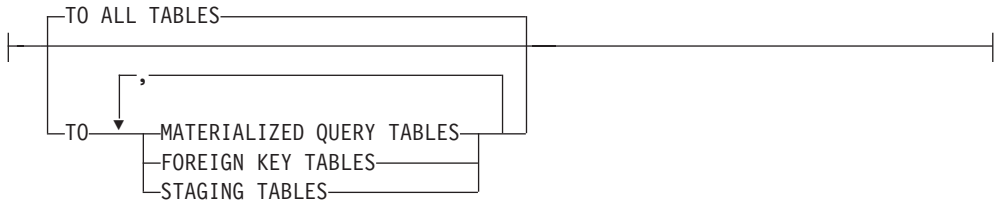
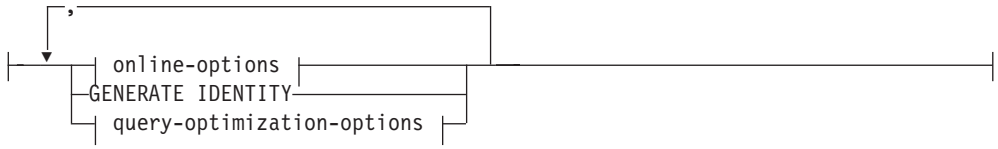


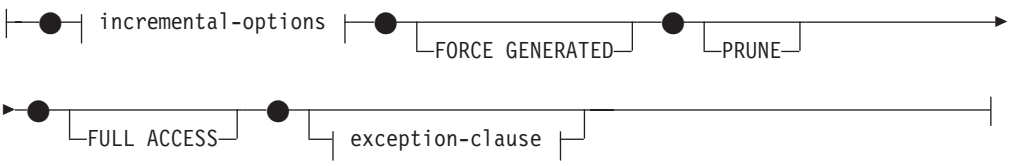
table-checked-options:



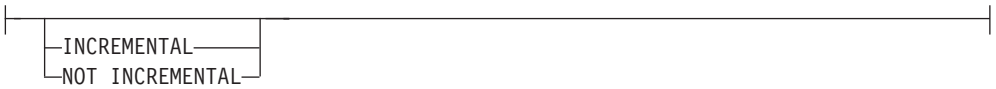
online-options:

query-optimization-options:

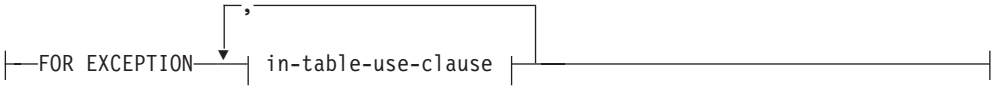
check-options:



incremental-options:

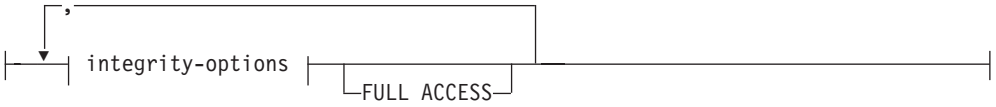


exception-clause:

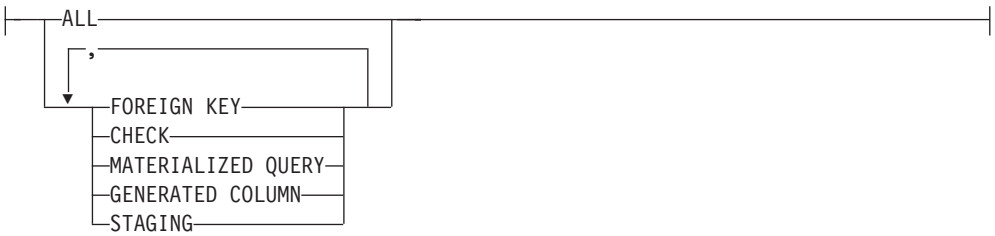


in-table-use-clause:

table-unchecked-options:



integrity-options:



Description

FOR *table-name*
Identifies one or more tables for integrity processing. It must be a table described in the catalog and must not be a view, catalog table, or typed table.

OFF
Specifies that the tables are placed in set integrity pending state. Only very limited activity is allowed on a table that is in set integrity pending state.

access-mode-clause
Specifies the readability of the table while it is in set integrity pending state.

NO ACCESS
Specifies that the table is to be put in set integrity pending no access state, which does not allow read or write access to the table.

READ ACCESS
Specifies that the table is to be put in set integrity pending read access state, which allows read access to the non-appended portion of the table. This option is not allowed on a table that is in set integrity pending no access state (SQLSTATE 428FH).

cascade-clause
Specifies whether the set integrity pending state of the table referenced in the SET INTEGRITY statement is to be immediately cascaded to descendent tables.

CASCADE IMMEDIATE
Specifies that the set integrity pending state is to be immediately extended to descendent tables.

to-descendent-types
Specifies the type of descendent tables to which the set integrity pending state is immediately cascaded.

TO ALL TABLES
Specifies that the set integrity pending state is to be immediately cascaded to all descendent tables of the tables in the invocation list. Descendent tables include all descendent foreign key tables, immediate staging tables, and immediate materialized query tables that are descendants of the tables in the invocation list, or descendants of descendent foreign key tables.

Specifying TO ALL TABLES is equivalent to specifying TO FOREIGN KEY TABLES, TO MATERIALIZED QUERY TABLES, and TO STAGING TABLES, all in the same statement.

TO MATERIALIZED QUERY TABLES
If only TO MATERIALIZED QUERY TABLES is specified, the set integrity pending state is to be immediately cascaded only to descendent immediate materialized query tables. Other descendent tables might later be put in set integrity pending state, if necessary,

when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO MATERIALIZED QUERY TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate materialized query tables of the tables in the invocation list, and to all immediate materialized query tables that are descendants of the descendent foreign key tables.

TO FOREIGN KEY TABLES
Specifies that the set integrity pending state is to be immediately cascaded to descendent foreign key tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state.

TO STAGING TABLES
Specifies that the set integrity pending state is to be immediately cascaded to descendent staging tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO STAGING TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate staging tables of the tables in the invocation list, and to all immediate staging tables that are descendants of the descendent foreign key tables.

CASCADE DEFERRED
Specifies that only the tables in the invocation list are to be put in set integrity pending state. The states of the descendent tables will remain unchanged. Descendent foreign key tables might later be implicitly put in set integrity pending state when their parent tables are checked for constraints violations. Descendent immediate materialized query tables and descendent immediate staging tables might be implicitly put in set integrity pending state when one of their underlying tables is checked for integrity violations. A query of a table that is in the set integrity pending state might succeed if an eligible materialized query table that is not in the set integrity pending state is accessed by the query instead of the specified table.

If *cascade-clause* is not specified, the set integrity pending state is immediately cascaded to all descendent tables.

IMMEDIATE CHECKED
Specifies that the table is to be taken out of set integrity pending state by performing required integrity processing on the table. This is done in accordance with the information set in the STATUS and CONST_CHECKED columns of the SYSCAT.TABLES catalog view. That is:

- The value in the STATUS column must be 'C' (the table is in set integrity pending state), or an error is returned (SQLSTATE 51027), unless the table is a descendent foreign key table, descendent materialized query table, or descendent staging table of a table that is specified in the list, is in set integrity pending state, and whose intermediate ancestors are also in the list.
- If the table being checked is in set integrity pending state, the value in CONST_CHECKED indicates which integrity options are to be checked.

When the table is taken out of set integrity pending state, its descendent tables are, if necessary, put in set integrity pending state. A warning to indicate that descendent tables have been put in set integrity pending state is returned (SQLSTATE 01586).

If the table is a system-maintained materialized query table, the data is checked against the query and refreshed as necessary. (IMMEDIATE CHECKED cannot be used for user-maintained materialized query tables.) If the table is a staging table, the data is checked against its query definition and propagated as necessary.

When the integrity of a child table is checked:

- None of its parents can be in set integrity pending state, or
- Each of its parents must be checked for constraints violations in the same SET INTEGRITY statement

When an immediate materialized query table is refreshed, or deltas are propagated to a staging table:

- None of its underlying tables can be in set integrity pending state, or
- Each of its underlying tables must be checked in the same SET INTEGRITY statement

Otherwise, an error is returned (SQLSTATE 428A8).

table-checked-options

online-options
Specifies the accessibility of the table while it is being processed.

ALLOW NO ACCESS
Specifies that no other users can access the table while it is being processed, except if they are using the Uncommitted Read isolation level.

ALLOW READ ACCESS
Specifies that other users have read-only access to the table while it is being processed.

ALLOW WRITE ACCESS
Specifies that other users have read and write access to the table while it is being processed.

GENERATE IDENTITY
Specifies that if the table includes an identity column, the values are generated by the SET INTEGRITY statement. By default, when the GENERATE IDENTITY option is specified, only attached rows will have their identity column values generated by the SET INTEGRITY statement. The NOT INCREMENTAL option must be specified in conjunction with the GENERATE IDENTITY option to have the SET INTEGRITY statement generate identity column values for all rows in the table, including attached rows, loaded rows, and existing rows. If the GENERATE IDENTITY option is not specified, the current identity column values for all rows in the table are left unchanged. When the table is a system-period temporal table, GENERATE IDENTITY with the NOT INCREMENTAL option is allowed only if you first issue an ALTER TABLE statement with the DROP VERSIONING clause (SQLSTATE 428FH).

query-optimization-options
Specifies the query optimization options for the maintenance of REFRESH DEFERRED materialized query tables.

ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY
Specifies that when the CURRENT REFRESH AGE special register

is set to 'ANY', the maintenance of *table-name* will allow REFRESH DEFERRED materialized query tables to be used to optimize the query that maintains *table-name*. If *table-name* is not a REFRESH DEFERRED materialized query table, an error is returned (SQLSTATE 428FH). REFRESH IMMEDIATE materialized query tables are always considered during query optimization.

check-options

incremental-options

INCREMENTAL
Specifies the application of integrity processing on the appended portion (if any) of the table. If such a request cannot be satisfied (that is, the system detects that the whole table needs to be checked for data integrity), an error is returned (SQLSTATE 55019).

NOT INCREMENTAL
Specifies the application of integrity processing on the whole table. If the table is a materialized query table, the materialized query table definition is recomputed. If the table has at least one constraint defined on it, this option causes full processing of descendent foreign key tables and descendent immediate materialized query tables. If the table is a staging table, it is set to an inconsistent state.

If the *incremental-options* clause is not specified, the system determines whether incremental processing is possible; if not, the whole table is checked.

FORCE GENERATED
If the table includes generated by expression columns, the values are computed on the basis of the expression and stored in the column. If this option is not specified, the current values are compared to the computed value of the expression, as though an equality check constraint were in effect. If the table is processed for integrity incrementally, generated columns are computed only for the appended portion. When the table is a system-period temporal table, the FORCE GENERATED option is allowed only if you first issue an ALTER TABLE statement with the DROP VERSIONING clause (SQLSTATE 428FH).

PRUNE
This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH). If the INCREMENTAL check option is also specified, an error is returned (SQLSTATE 428FH).

FULL ACCESS
Specifies that the table is to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table (that has dependent immediate materialized query tables or dependent immediate staging tables) in the invocation list is incrementally processed, the underlying table is put in no data movement state, as required, after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables are taken out of set

integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE CHECKED option, the underlying table is put directly in full access state (bypassing the no data movement state). In DB2 Version 9.7. Fix Pack 1 and later, specifying the FULL ACCESS option only removes the dependency between the dependent tables and underlying table. The underlying table continues to be unavailable until the data partition detach process is completed by the asynchronous partition detach task.

Dependent immediate materialized query tables that have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them might be flagged as inconsistent.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is put directly into full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option was specified.

exception-clause

FOR EXCEPTION
Specifies that any row that is in violation of a constraint being checked is to be moved to an exception table. Even if errors are detected, the table is taken out of set integrity pending state. A warning to indicate that one or more rows have been moved to the exception tables is returned (SQLSTATE 01603).

If the FOR EXCEPTION option is not specified and any constraints are violated, only the first detected violation is returned (SQLSTATE 23514). If there is a violation in any table, all of the tables are left in set integrity pending state.

It is recommended to always use the FOR EXCEPTION option when checking for constraints violations to prevent a rollback of the SET INTEGRITY statement if a violation is found.

When the table specified after the IN keyword is a system-period temporal table, the FOR EXCEPTION option is allowed only if you first issue an ALTER TABLE statement with the DROP VERSIONING clause (SQLSTATE 428FH).

IN *table-name*
Specifies the table from which rows that violate constraints are to be moved. There must be one exception table specified for each table being checked. This clause cannot be specified for a materialized query table or a staging table (SQLSTATE 428A7).

USE *table-name*
Specifies the exception table into which error rows are to be moved.

FULL ACCESS
If the FULL ACCESS option is specified as the only operation of the statement, the table is placed into the full access state without being rechecked for integrity violations. However, dependent immediate materialized query tables that have not been refreshed might require a full recomputation in subsequent

REFRESH TABLE statements, and dependent immediate staging tables that have not had the delta portions of the table propagated to them might be changed to incomplete state. This option can only be specified for a table that is in the no data movement state or the no access state, but not in the set integrity pending state (SQLSTATE 428FH).

PRUNE
This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH).

table-unchecked-options

integrity-options
Used to define the types of required integrity processing that are to be bypassed when the table is taken out of the set integrity pending state.

ALL
The table will be immediately taken out of set integrity pending state without any of its required integrity processing being performed.

FOREIGN KEY
Required foreign key constraints checking will not be performed when the table is brought out of set integrity pending state.

CHECK
Required check constraints checking will not be performed when the table is brought out of set integrity pending state.

MATERIALIZED QUERY
Required refreshing of a materialized query table will not be performed when the table is brought out of set integrity pending state.

GENERATED COLUMN
Required generated column constraints checking will not be performed when the table is brought out of set integrity pending state.

STAGING
Required propagation of data to a staging table will not be performed when the table is brought out of set integrity pending state.

If no other types of integrity processing are required on the table after a specific type of integrity processing has been marked as bypassed, the table is immediately taken out of set integrity pending state.

FULL ACCESS
Specifies that the tables are to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table in the invocation list is incrementally processed, and it has dependent immediate materialized query tables or dependent immediate staging tables, the underlying table is placed, as required, in the no data movement state after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables have been taken out of set integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE UNCHECKED option, the underlying table is placed directly in full access state (it bypasses the no data movement state). Dependent immediate materialized query tables that

have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them might be flagged as inconsistent.

In DB2 V9.7. Fix Pack 1 and later, specifying the FULL ACCESS option only removes the dependency between the dependent tables and underlying table. The underlying table continues to be unavailable until the data partition detach process is completed by the asynchronous partition detach task.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is placed directly in full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option has been specified.

If the FULL ACCESS option has been specified with the IMMEDIATE UNCHECKED option, and the statement does not bring the table out of set integrity pending state, an error is returned (SQLSTATE 428FH).

IMMEDIATE UNCHECKED

Specifies one of the following:

- The table is to be brought out of set integrity pending state immediately without any required integrity processing.
- The table is to have one or more types of required integrity processing bypassed when the table is brought out of set integrity pending state by a subsequent SET INTEGRITY statement using the IMMEDIATE CHECKED option.

Consider the data integrity implications of this option before using it. See the "Notes" section.

Notes

- Effects on tables in one of the restricted set integrity-related states:
  - Use of INSERT, UPDATE, or DELETE is disallowed on a table that is in read access state or in no access state. Furthermore, any statement that requires this type of modification to a table that is in such a state will be rejected. For example, deletion of a row in a parent table that cascades to a dependent table that is in the no access state is not allowed.
  - Use of SELECT is disallowed on a table that is in the no access state. Furthermore, any statement that requires read access to a table that is in the no access state will be rejected.
  - New constraints added to a table are normally enforced immediately. However, if the table is in set integrity pending state, the checking of any new constraints is deferred until the table is taken out of set integrity pending state. If the table is in set integrity pending state, addition of a new constraint places the table into set integrity pending no access state, because validity of data is at risk.
  - The CREATE INDEX statement cannot reference any table that is in read access state or in no access state. Similarly, an ALTER TABLE statement to add a primary key or a unique constraint cannot reference any table that is in read access state or in no access state.
  - The import utility is not allowed to operate on a table that is in read access state or in no access state.

  - The export utility is not allowed to operate on a table that is in no access state, but is allowed to operate on a table that is in read access state. If a table is in read access state, the export utility will only export the data that is in the non-appended portion.
  - Operations (like REORG, REDISTRIBUTE, update distribution key, update multidimensional clustering key, update range clustering key, update table partitioning key, and so on) that might involve data movement within a table are not allowed on a table that is in any of the following states: read access, no access, or no data movement.
  - The load, backup, restore, update statistics, runstats, reorgchk, list history, and rollforward utilities are allowed on a table that is in any of the following states: full access, read access, no access, or no data movement.
  - The ALTER TABLE, COMMENT, DROP TABLE, CREATE ALIAS, CREATE TRIGGER, CREATE VIEW, GRANT, REVOKE, and SET INTEGRITY statements can reference a table that is in any of the following states: full access, read access, no access, or no data movement. However, they might cause the table to be put into no access state.
  - Packages, views, and any other objects that depend on a table that is in no access state will return an error when the table is accessed at run time. Packages that depend on a table that is in read access state will return an error when an insert, update, or delete operation is attempted on the table at run time.

  The removal of violating rows by the SET INTEGRITY statement is not a delete event. Therefore, triggers are never activated by a SET INTEGRITY statement. Similarly, updating generated columns using the FORCE GENERATED option does not activate triggers.
- Warning about the use of the IMMEDIATE UNCHECKED clause:
  - This clause is intended to be used by utility programs, and its use by application programs is not recommended. If there is data in the table that does not meet the integrity specifications that were defined for the table, and the IMMEDIATE UNCHECKED option is used, incorrect query results might be returned.

    The fact that the table was taken out of the set integrity pending state without performing the required integrity processing will be recorded in the catalog (the respective byte in the CONST_CHECKED column in the SYSCAT.TABLES view will be set to 'U'). This indicates that the user has assumed responsibility for data integrity with respect to the specific constraints. This value remains unchanged until either:
    - The table is put back into set integrity pending state (by referencing the table in a SET INTEGRITY statement with the OFF option), at which time 'U' values in the CONST_CHECKED column are changed to 'W' values, indicating that the user had previously assumed responsibility for data integrity, and the system needs to verify the data.
    - All unchecked constraints for the table are dropped.

    The 'W' state differs from the 'N' state in that it records the fact that integrity was previously checked by the user, but not yet by the system. If the user issues the SET INTEGRITY ... IMMEDIATE CHECKED statement with the NOT INCREMENTAL option, the system rechecks the whole table for data integrity (or performs a full refresh on a materialized query table), and then changes the 'W' state to the 'Y' state. If IMMEDIATE UNCHECKED is specified, or if NOT INCREMENTAL is not specified, the 'W' state is changed back to the 'U' state to record the fact that some data has still not been

verified by the system. In the latter case (when the NOT INCREMENTAL is not specified), a warning is returned (SQLSTATE 01636).

If an underlying table's integrity has been checked using the IMMEDIATE UNCHECKED clause, the 'U' values in the CONST_CHECKED column of the underlying table will be propagated to the corresponding CONST_CHECKED column of:

- Dependent immediate materialized query tables
- Dependent deferred materialized query tables
- Dependent staging tables

For a dependent immediate materialized query table, this propagation is done whenever the underlying table is brought out of set integrity pending state, and whenever the materialized query table is refreshed. For a dependent deferred materialized query table, this propagation is done whenever the materialized query table is refreshed. For dependent staging tables, this propagation is done whenever the underlying table is brought out of set integrity pending state. These propagated 'U' values in the CONST_CHECKED columns of dependent materialized query tables and staging tables record the fact that these materialized query tables and staging tables depend on some underlying table whose required integrity processing has been bypassed using the IMMEDIATE UNCHECKED option.

For a materialized query table, the 'U' value in the CONST_CHECKED column that was propagated by the underlying table will remain until the materialized query table is fully refreshed and none of its underlying tables have a 'U' value in their corresponding CONST_CHECKED column. After such a refresh, the 'U' value in the CONST_CHECKED column for the materialized query table will be changed to 'Y'.

For a staging table, the 'U' value in the CONST_CHECKED column that was propagated by the underlying table will remain until the corresponding deferred materialized query table of the staging table is refreshed. After such a refresh, the 'U' value in the CONST_CHECKED column for the staging table will be changed to 'Y'.

– If a child table and its parent table are checked in the same SET INTEGRITY statement with the IMMEDIATE CHECKED option, and the parent table requires full checking of its constraints, the child table will have its foreign key constraints checked, independently of whether or not the child table has a 'U' value in the CONST_CHECKED column for foreign key constraints.

- If the table is data partitioned and there are nonpartitioned indexes (except the XML column path index) to maintain, IMMEDIATE UNCHECKED behavior when a single target table is specified is the same as IMMEDIATE CHECKED behavior with the ALLOW WRITE ACCESS option: all integrity processing is performed and any resulting errors are returned. If the statement references more than one target table, an error is returned (SQLSTATE 428FH).
- After appending data using LOAD INSERT or ALTER TABLE ATTACH, the SET INTEGRITY statement with the IMMEDIATE CHECKED option checks the table for constraints violations. The system determines whether incremental processing on the table is possible. If so, only the appended portion is checked for integrity violations. If not, the system checks the whole table for integrity violations.
- Consider the statement:

```
SET INTEGRITY FOR T IMMEDIATE CHECKED
```

In the following scenarios, neither the INCREMENTAL check option for T nor an incremental refresh of T---if T is a materialized query table (MQT) or a staging table---is supported:
  - New constraints have been added to T while it is in set integrity pending state
  - When a LOAD REPLACE operation against T, it parents, or its underlying tables has taken place
  - When the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated after the last integrity check on T, its parents, or its underlying tables
  - The cascading effect of full processing, when any parent of T (or underlying table, if T is a materialized query table or a staging table) has been checked for integrity non-incrementally
  - If the table space containing the table or its parent (or underlying table of a materialized query table or a staging table) has been rolled forward to a point in time, and the table and its parent (or underlying table if the table is a materialized query table or a staging table) reside in different table spaces
  - T is an MQT, and a LOAD REPLACE or LOAD INSERT operation directly into T has taken place after the last refresh
- Incremental processing will be used whenever the situation allows it, because it is more efficient. The INCREMENTAL option is not needed in most cases. It is needed, however, to ensure that integrity checks are indeed processed incrementally. If the system detects that full processing is needed to ensure data integrity, an error is returned (SQLSTATE 55019).
- If the conditions for full processing described in the previous bullet are not satisfied, the system will attempt to check only the appended portion for integrity, or perform an incremental refresh (if it is a materialized query table) when the user does not specify the NOT INCREMENTAL option for the statement `SET INTEGRITY FOR T IMMEDIATE CHECKED`.
- If an error occurs during integrity processing, all the effects of the processing (including deleting from the original and inserting into the exception tables) will be rolled back.
- If a SET INTEGRITY statement issued with the FORCE GENERATED option fails because of a lack of log space, increase available active log space and reissue the SET INTEGRITY statement. Alternatively, use the SET INTEGRITY statement with the GENERATED COLUMN and IMMEDIATE UNCHECKED options to bypass generated column checking for the table. Then, issue a SET INTEGRITY statement with the IMMEDIATE CHECKED option and without the FORCE GENERATED option to check the table for other integrity violations (if applicable) and to bring it out of set integrity pending state. After the table is out of the set integrity pending state, the generated columns can be updated to their default (generated) values by assigning them to the keyword DEFAULT in an UPDATE statement. This is accomplished by using either multiple searched update statements based on ranges (each followed by a commit), or a cursor-based approach using intermittent commits. A “with hold” cursor should be used if locks are to be retained after intermittent commits using the cursor-based approach.
- A table that was put into set integrity pending state using the CASCADE DEFERRED option of the SET INTEGRITY statement or the LOAD command, or through the ALTER TABLE statement with the ATTACH clause, and that is checked for integrity violations using the IMMEDIATE CHECKED option of the SET INTEGRITY statement, will have its descendent foreign key tables,

descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:
  - If the entire table is checked for integrity violations, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  - If the table is checked for integrity violations incrementally, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  - If the table requires no checking at all, its descendent immediate materialized query tables, descendent staging tables, and descendent foreign key tables will remain in their original states.
- A table that was put in set integrity pending state using the CASCADE DEFERRED option (of the SET INTEGRITY statement or the LOAD command), and that is brought out of set integrity pending state using the IMMEDIATE UNCHECKED option of the SET INTEGRITY statement, will have its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:
  - If the table has been loaded using the REPLACE mode, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  - If the table has been loaded using the INSERT mode, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  - If the table has not been loaded, its descendent immediate materialized query tables, descendent staging tables, and its descendent foreign key tables will remain in their original states.
- SET INTEGRITY is usually a long running statement. In light of this, to reduce the risk of a rollback of the entire statement because of a lock timeout, you can issue the SET CURRENT LOCK TIMEOUT statement with the WAIT option before executing the SET INTEGRITY statement, and then reset the special register to its previous value after the transaction commits. Note, however, that the CURRENT LOCK TIMEOUT special register only impacts a specific set of lock types.
- If you use the ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY option, ensure that the maintenance order is correct for REFRESH DEFERRED materialized query tables. For example, consider two materialized query tables, MQT1 and MQT2, whose materialized queries share the same underlying tables. The materialized query for MQT2 can be calculated using MQT1, instead of the underlying tables. If separate statements are used to maintain these two materialized query tables, and MQT2 is maintained first, the system might choose to use the contents of MQT1, which has not yet been maintained, to maintain MQT2. In this case, MQT1 would contain current data, but MQT2 could still contain stale data, even though both were maintained at almost the same time. The correct maintenance order, if two SET INTEGRITY statements are used instead of one, is to maintain MQT1 first.
- When using the SET INTEGRITY statement to perform integrity processing on a base table that has been loaded or attached, it is recommended that you process its dependent REFRESH IMMEDIATE materialized query tables and its PROPAGATE IMMEDIATE staging tables in the same SET INTEGRITY statement to avoid putting these dependent tables in set integrity pending no

access state at the end of SET INTEGRITY processing. Note that for base tables that have a large number of dependent REFRESH IMMEDIATE materialized query tables and PROPAGATE IMMEDIATE staging tables, memory constraints might make it impossible to process all of the dependents in the same statement as the base table.

- If the FORCE GENERATED or the GENERATE IDENTITY option is specified, and the column that is generated is part of a unique index, the SET INTEGRITY statement returns an error (SQLSTATE 23505) and rolls back if it detects duplicate keys in the unique index. This error is returned even if there is an exception table for the table being processed.

  This scenario can occur under the following circumstances:

  – The SET INTEGRITY statement runs after a LOAD command against the table, and the GENERATEDOVERRIDE or the IDENTITYOVERRIDE file type modifier is specified during the load operation. To prevent this scenario, it is recommended that you use the GENERATEDIGNORE or the GENERATEDMISSING file type modifier instead of GENERATEDOVERRIDE, and that you use the IDENTITYIGNORE or the IDENTITYMISSING modifier instead of IDENTITYOVERRIDE. Using the recommended modifiers will prevent the need for any generated by expression column or identity column processing during SET INTEGRITY statement execution.
  – The SET INTEGRITY statement is run after an ALTER TABLE statement that alters the expression of a generated by expression column.

  To bring a table out of the set integrity pending state after encountering such a scenario:

  – Do not use the FORCE GENERATED or the GENERATE IDENTITY option to regenerate the column values. Instead, use the IMMEDIATE CHECKED option in conjunction with the FOR EXCEPTION option to move any rows that violate the generated column expression to an exception table. Then, re-insert the rows into the table from the exception table, which will generate the correct expression and perform unique key checking. This prevents having to reprocess the entire table, because only those rows that violated the generated column expression will need to be processed again.
  – If the table being processed has attached partitions, detach those partitions before performing the actions that are described in the previous bullet. Then, re-attach the partitions and execute a SET INTEGRITY statement to process integrity on the attached partitions separately.
- If a protected table is specified for the SET INTEGRITY statement along with an exception table, all of the following table criteria must be met; otherwise, an error is returned (SQLSTATE 428A5):

  – The tables must be protected by the same security policy.
  – If a column in the protected table has data type DB2SECURITYLABEL, the corresponding column in the exception table must also have data type DB2SECURITYLABEL.
  – If a column in the protected table is protected by a security label, the corresponding column in the exception table must also be protected by the same security label.
- Rows that violate the integrity being checked in a system-period temporal table cannot be moved to an exception table. If the violating rows must be moved to an exception table, the table must be altered to drop versioning before issuing the SET INTEGRITY statement with the FOR EXCEPTION clause.
- ***Syntax alternatives***: The following are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.

- SET CONSTRAINTS can be specified in place of SET INTEGRITY
- SUMMARY can be specified in place of MATERIALIZED QUERY

Examples

- *Example 1:* The following is an example of a query that provides information about the set integrity pending state and the set integrity-related access restriction states of tables. SUBSTR is used to extract individual bytes of the CONST_CHECKED column of SYSCAT.TABLES. The first byte represents foreign key constraints; the second byte represents check constraints; the fifth byte represents materialized query table integrity; the sixth byte represents generated column constraints; the seventh byte represents staging table integrity; and the eighth byte represents data partitioning constraints. STATUS gives the set integrity pending state, and ACCESS_MODE gives the set integrity-related access restriction state.

```
SELECT TABNAME, STATUS, ACCESS_MODE,
  SUBSTR(CONST_CHECKED,1,1) AS FK_CHECKED,
  SUBSTR(CONST_CHECKED,2,1) AS CC_CHECKED,
  SUBSTR(CONST_CHECKED,5,1) AS MQT_CHECKED,
  SUBSTR(CONST_CHECKED,6,1) AS GC_CHECKED,
  SUBSTR(CONST_CHECKED,7,1) AS STG_CHECKED,
  SUBSTR(CONST_CHECKED,8,1) AS DP_CHECKED
FROM SYSCAT.TABLES
```

- *Example 2:* Put the PARENT table in set integrity pending no access state, and immediately cascade the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
  NO ACCESS CASCADE IMMEDIATE
```

- *Example 3:* Put the PARENT table in set integrity pending read access state without immediately cascading the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
  READ ACCESS CASCADE DEFERRED
```

- *Example 4:* Check integrity for a table named FACT_TABLE. If there are no integrity violations detected, the table is brought out of set integrity pending state. If any integrity violations are detected, the entire statement is rolled back, and the table remains in set integrity pending state.

```
SET INTEGRITY FOR FACT_TABLE IMMEDIATE CHECKED
```

- *Example 5:* Check integrity for the SALES and PRODUCTS tables, and move the rows that violate integrity into exception tables named SALES_EXCEPTIONS and PRODUCTS_EXCEPTIONS. Both the SALES and PRODUCTS tables are brought out of set integrity pending state, whether or not there are any integrity violations.

```
SET INTEGRITY FOR SALES, PRODUCTS IMMEDIATE CHECKED
  FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS,
  IN PRODUCTS USE PRODUCTS_EXCEPTIONS
```

- *Example 6:* Enable FOREIGN KEY constraint checking in the MANAGER table, and CHECK constraint checking in the EMPLOYEE table, to be bypassed with the IMMEDIATE UNCHECKED option.

```
SET INTEGRITY FOR MANAGER FOREIGN KEY,
  EMPLOYEE CHECK IMMEDIATE UNCHECKED
```

- *Example 7:* Add a check constraint and a foreign key to the EMP_ACT table, using two ALTER TABLE statements. The SET INTEGRITY statement with the OFF option is used to put the table in set integrity pending state, so that the constraints are not checked immediately upon execution of the two ALTER TABLE statements. The single SET INTEGRITY statement with the IMMEDIATE CHECKED option is used to check both of the added constraints during a single pass through the table.

```
SET INTEGRITY FOR EMP_ACT OFF;
ALTER TABLE EMP_ACT ADD CHECK
  (EMSTDATE <= EMENDATE);
ALTER TABLE EMP_ACT ADD FOREIGN KEY
  (EMPNO) REFERENCES EMPLOYEE;
SET INTEGRITY FOR EMP_ACT IMMEDIATE CHECKED
  FOR EXCEPTION IN EMP_ACT USE EMP_ACT_EXCEPTIONS
```

- *Example 8:* Update generated columns with the correct values.

```
SET INTEGRITY FOR SALES IMMEDIATE CHECKED
  FORCE GENERATED
```

- *Example 9:* Append (using LOAD INSERT) from different sources into an underlying table (SALES) of a REFRESH IMMEDIATE materialized query table (SALES_SUMMARY). Check SALES incrementally for data integrity, and refresh SALES_SUMMARY incrementally. In this scenario, integrity checking for SALES and refreshing of SALES_SUMMARY are incremental, because the system chooses incremental processing. The ALLOW READ ACCESS option is used on the SALES table to allow concurrent reads of existing data while integrity checking of the loaded portion of the table is taking place.

```
LOAD FROM 2000_DATA.DEL OF DEL
  INSERT INTO SALES ALLOW READ ACCESS;
LOAD FROM 2001_DATA.DEL OF DEL
  INSERT INTO SALES ALLOW READ ACCESS;
SET INTEGRITY FOR SALES ALLOW READ ACCESS IMMEDIATE CHECKED
  FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS;
REFRESH TABLE SALES_SUMMARY;
```

- *Example 10:* Attach a new partition to a data partitioned table named SALES. Incrementally check for constraints violations in the attached data of the SALES table and incrementally refresh the dependent SALES_SUMMARY table. The ALLOW WRITE ACCESS option is used on both tables to allow concurrent updates while integrity checking is taking place.

```
ALTER TABLE SALES
  ATTACH PARTITION STARTING (100) ENDING (200)
  FROM SOURCE;
SET INTEGRITY FOR SALES ALLOW WRITE ACCESS, SALES_SUMMARY ALLOW WRITE ACCESS
  IMMEDIATE CHECKED FOR EXCEPTION IN SALES
  USE SALES_EXCEPTIONS;
```

- *Example 11:* Detach a partition from a data partitioned table named SALES. Incrementally refresh the dependent SALES_SUMMARY table.

```
ALTER TABLE SALES
  DETACH PARTITION 2000_PART INTO ARCHIVE_TABLE;
SET INTEGRITY FOR SALES_SUMMARY
  IMMEDIATE CHECKED;
```

- *Example 12:* Bring a new user-maintained materialized query table out of set integrity pending state.

```
CREATE TABLE YEARLY_SALES
  AS (SELECT YEAR, SUM(SALES)AS SALES
  FROM FACT_TABLE GROUP BY YEAR)
  DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY USER

SET INTEGRITY FOR YEARLY_SALES
  ALL IMMEDIATE UNCHECKED
```

- *Example 13:* Attach a new partition to a data partitioned table named SALES. Assume that this table has no nonpartitioned user indexes. Assume also that data integrity checking, including range validation and other constraints checking, has already been done (through application logic that is independent of the data server). Optimize the data roll-in process by using the SET INTEGRITY...ALL IMMEDIATE UNCHECKED statement to skip range and constraints violation checking.

```
ALTER TABLE SALES
  ATTACH PARTITION STARTING (300) ENDING (400)
  FROM SOURCE_TABLE;
SET INTEGRITY FOR SALES ALL IMMEDIATE UNCHECKED;
```

The SALES table is brought out of SET INTEGRITY pending state, and the new data is available for applications to use immediately.

SET PASSTHRU

The SET PASSTHRU statement opens and closes a session for submitting a data source's native SQL directly to that data source.

The statement is not under transaction control.

Invocation

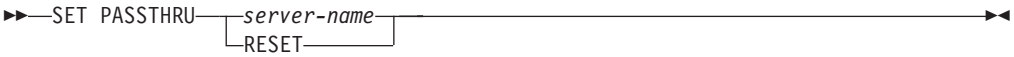
This statement can be issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must provide authorization to:

- Pass through to the data source
- Satisfy security measures at the data source

Syntax



Description

server-name
Names the data source for which a pass-through session is to be opened. *server-name* must identify a data source that is described in the catalog.

RESET
Closes a pass-through session.

Notes

- The following restrictions apply to Microsoft SQL Server, Sybase, and Oracle data sources:
  - User-defined transactions cannot be used for Microsoft SQL Server and Sybase data sources in pass-through mode, because Microsoft SQL Server and Sybase restrict which SQL statements can be specified within a user-defined transaction. Because SQL statements that are processed in pass-through mode are not parsed by DB2, it is not possible to detect whether the user specified an SQL statement that is permitted within a user-defined transaction.
  - The COMPUTE clause is not supported on Microsoft SQL Server and Sybase data sources.
  - DDL statements are not subject to transaction semantics on Microsoft SQL Server, Oracle and Sybase data sources. The operation, when complete, is automatically committed by Microsoft SQL Server, Oracle or Sybase. If a rollback occurs, the DDL is not rolled back.

Examples

- *Example 1:* Start a pass-through session to data source BACKEND.

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
```

- *Example 2:* Start a pass-through session with a PREPARE statement.

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL PREPARE STMT FROM :PASS_THRU;
EXEC SQL EXECUTE STMT;
```

- *Example 3:* End a pass-through session.

```
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

- *Example 4:* Use the PREPARE and EXECUTE statements to end a pass-through session.

```
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL PREPARE STMT FROM :PASS_THRU_RESET;
EXEC SQL EXECUTE STMT;
```

- *Example 5:* Open a session to pass through to a data source, create a clustered index for a table at this data source, and close the pass-through session.

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
EXEC SQL PREPARE STMT                 pass-through mode
  FROM "CREATE UNIQUE
        CLUSTERED INDEX TABLE_INDEX
        ON USER2.TABLE               table is not an
        WITH IGNORE DUP KEY";         alias
EXEC SQL EXECUTE STMT;
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

SET PATH

The SET PATH statement changes the value of the CURRENT PATH special register.

This statement is not under transaction control.

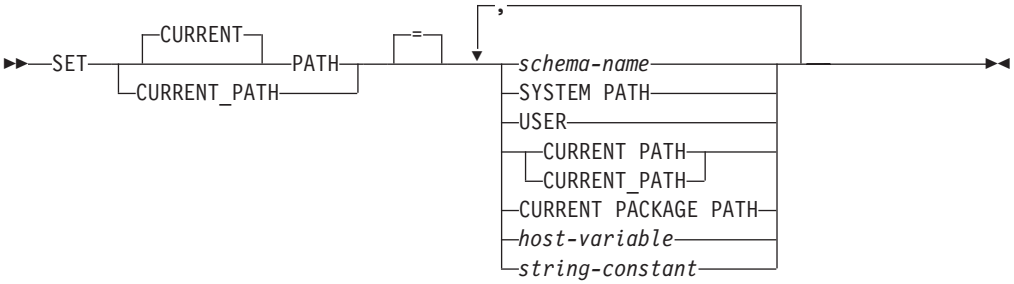
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

schema-name
This one-part name identifies a schema that exists at the application server. No validation that the schema exists is made at the time that the path is set. If a *schema-name* is, for example, misspelled, the error will not be caught, and it could affect the way subsequent SQL operates.

SYSTEM PATH
This value is the same as specifying the schema names *"SYSIBM","SYSFUN","SYSPROC","SYSIBMADM"*.

USER
The value of the USER special register.

CURRENT PATH
The value of the CURRENT PATH special register before this statement executes.

CURRENT PACKAGE PATH
The value of the CURRENT PACKAGE PATH special register.

host-variable
A variable of type CHAR or VARCHAR. The length of the contents of the *host-variable* must not exceed 128 bytes (SQLSTATE 42815). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of the *host-variable* must be left-aligned. When specifying the *schema-name* with a *host-variable*, all characters must be specified in the exact case intended as there is no conversion to uppercase characters.

string-constant
A character string constant with a maximum length of 128 bytes.

Rules

- A schema name cannot appear more than once in the SQL path (SQLSTATE 42732).
- The schema name SYSPUBLIC cannot be specified in the SQL path (SQLSTATE 42815).
- The number of schemas that can be specified is limited by the total length of the CURRENT PATH special register. The special register string is built by taking each schema name specified and removing trailing blanks, delimiting with double quotation marks, doubling quotation marks within the schema name as necessary, and then separating each schema name by a comma. The length of the resulting string cannot exceed 2048 bytes (SQLSTATE 42907).

Notes

- The initial value of the CURRENT PATH special register is "SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","X" where X is the value of the USER special register.
- The schema SYSIBM does not need to be specified. If it is not included in the SQL path, it is implicitly assumed as the first schema (in this case, it is not included in the CURRENT PATH special register).
- The CURRENT PATH special register specifies the SQL path used to resolve function names, procedure names, data type names, global variable names, and module object names in dynamic SQL statements. The FUNCPATH bind option specifies the SQL path to be used for resolving function names, procedure names, data type names, global variable names, and module object names in static SQL statements.
- ***Syntax alternatives***: The following syntax alternatives are supported for compatibility with previous versions of DB2 and with other database products. These alternatives are non-standard and should not be used.
  – CURRENT FUNCTION PATH can be specified in place of CURRENT PATH

Examples

- *Example 1:* The following statement sets the CURRENT PATH special register.

```
SET PATH = FERMAT, "McDrw #8", SYSIBM
```

- *Example 2:* The following example retrieves the current value of the CURRENT PATH special register into the host variable called CURPATH.

```
EXEC SQL VALUES (CURRENT PATH) INTO :CURPATH;
```

The value would be "FERMAT","McDrw #8","SYSIBM" if set by the previous example.

SET ROLE

The SET ROLE statement verifies that the authorization ID of the session is a member of a specific role. An authorization ID acquires membership in a role when the role is granted to the authorization ID, or to a group or role in which the authorization ID is a member.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

None required.

Syntax

Description

role-name
Specifies a role in whose membership the authorization ID of the session is to be verified. The *role-name* must identify an existing role at the current server (SQLSTATE 42704). If the authorization ID of the session is not a member of *role-name*, an error is returned (SQLSTATE 42501).

Notes

- All roles that have been granted to an authorization ID are used for authorization checking. The SET ROLE statement does not affect which roles are used for this authorization checking. Use the GRANT ROLE and REVOKE ROLE statements to change the roles in which an authorization ID has membership.

Examples

- *Example 1:* User WALID has been granted the role EDITOR, but not the role AUTHOR. Verify that WALID is a member of the EDITOR role.

```
SET ROLE EDITOR
```

- *Example 2:* Verify that WALID is not a member of the AUTHOR role. The following statement returns an error (SQLSTATE 42501).

```
SET ROLE AUTHOR
```

SET SCHEMA

The SET SCHEMA statement changes the value of the CURRENT SCHEMA special register.

This statement is not under transaction control. If the package is bound with the DYNAMICRULES BIND option, this statement does not affect the qualifier used for unqualified database object references.

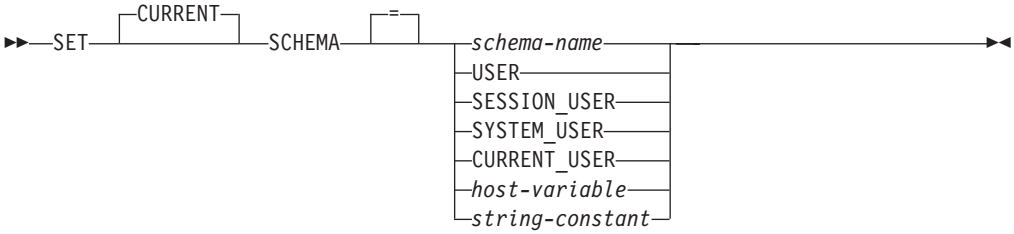
Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

schema-name
This one-part name identifies a schema that exists at the application server. The length must not exceed 128 bytes (SQLSTATE 42815). No validation that the schema exists is made at the time that the schema is set. If a *schema-name* is misspelled, the error will not be caught, and that could affect the way that subsequent SQL statements execute.

USER
The value in the USER special register.

SESSION_USER
The value in the SESSION_USER special register.

SYSTEM_USER
The value in the SYSTEM_USER special register.

CURRENT_USER
The value in the CURRENT_USER special register.

host-variable
A variable of type CHAR or VARCHAR. The length of the contents of the *host-variable* must not exceed 128 bytes (SQLSTATE 42815). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of the *host-variable* must be left-aligned. When specifying the *schema-name* with a *host-variable*, all characters must be specified in the exact case intended as there is no conversion to uppercase characters.

string-constant
A character string constant with a maximum length of 128 bytes.

Rules

- If the value specified does not conform to the rules for a *schema-name*, an error (SQLSTATE 3F000) is raised.
- The value of the CURRENT SCHEMA special register is used as the schema name in all dynamic SQL statements, with the exception of the CREATE SCHEMA statement, where an unqualified reference to a database object exists.
- The QUALIFIER bind option specifies the schema name for use as the qualifier for unqualified database object names in static SQL statements.

Notes

- The initial value of the CURRENT SCHEMA special register is equivalent to USER.
- Setting the CURRENT SCHEMA special register does not effect the CURRENT PATH special register. Hence, the CURRENT SCHEMA will not be included in the SQL path and functions, procedures and user-defined type resolution may not find these objects. To include the current schema value in the SQL path, whenever the SET SCHEMA statement is issued, also issue the SET PATH statement including the schema name from the SET SCHEMA statement.
- CURRENT SQLID is accepted as a synonym for CURRENT SCHEMA and the effect of a SET CURRENT SQLID statement will be identical to that of a SET CURRENT SCHEMA statement. No other effects, such as statement authorization changes, will occur.

Examples

- *Example 1:* The following statement sets the CURRENT SCHEMA special register.

```
SET SCHEMA RICK
```

- *Example 2:* The following example retrieves the current value of the CURRENT SCHEMA special register into the host variable called CURSCHEMA.
-

```
EXEC SQL VALUES (CURRENT SCHEMA) INTO :CURSCHEMA;
```

The value would be RICK, set by the previous example.

SET SERVER OPTION

The SET SERVER OPTION statement specifies a server option setting that is to remain in effect while a user or application is connected to the federated database. When the connection ends, this server option's previous setting is reinstated.

This statement is not under transaction control.

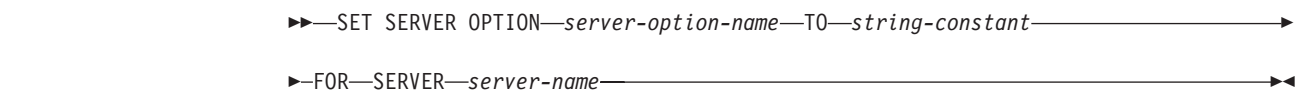
Invocation

This statement can be issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

None required.

Syntax



Description

server-option-name
Names the server option that is to be set.

TO *string-constant*
Specifies the setting for *server-option-name* as a character string constant.

SERVER *server-name*
Names the data source to which *server-option-name* applies. It must be a server described in the catalog.

Notes

- Server option names can be entered in uppercase or lowercase.
- One or more SET SERVER OPTION statements can be submitted when a user or application connects to the federated database. The statement (or statements) must be specified at the start of the first unit of work that is processed after the connection is established.
- SYSCAT.SERVEROPTIONS will not be updated based on a SET SERVER OPTION statement, because this change only affects the current connection.
- For static SQL, using the SET SERVER OPTION statement affects only the execution of the static SQL statement. Using the SET SERVER OPTION statement has no effect on the plans that are generated by the optimizer.

Examples

- *Example 1:* An Oracle data source called ORASERV is defined to a federated database called DJDB. ORASERV is configured to disallow plan hints. However, the DBA would like plan hints to be enabled for a test run of a new application. When the run is over, plan hints will be disallowed again.

```
CONNECT TO DJDB;
strcpy(stmt,"set server option plan_hints to 'Y' for server oraserv");
EXEC SQL EXECUTE IMMEDIATE :stmt;
strcpy(stmt,"select c1 from ora_t1 where c1 > 100"); /*Generate plan hints*/
```

```
EXEC SQL PREPARE s1 FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO :hv;
```

- *Example 2:* You have set the server option PASSWORD to 'Y' (validating passwords at the data source) for all Oracle 8 data sources. However, for a particular session in which an application is connected to the federated database in order to access a specific Oracle 8 data source-one defined to the federated database DJDB as ORA8A-passwords will not need to be validated.

```
CONNECT TO DJDB;
strcpy(stmt,"set server option password to 'N' for server ora8a");
EXEC SQL PREPARE STMT_NAME FROM :stmt;
EXEC SQL EXECUTE STMT_NAME FROM :stmt;
strcpy(stmt,"select max(c1) from ora8a_t1");
EXEC SQL PREPARE STMT_NAME FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR STMT_NAME;
EXEC SQL OPEN c1; /*Does not validate password at ora8a*/
EXEC SQL FETCH c1 INTO :hv;
```

SET SESSION AUTHORIZATION

The SET SESSION AUTHORIZATION statement changes the value of the SESSION_USER special register.

The statement is not under transaction control. The SET SESSION AUTHORIZATION statement is intended to provide support for a single user assuming different authorization IDs on the same connection, and should not be used for scenarios in which different users reuse the same connection, commonly referred to as connection pooling.

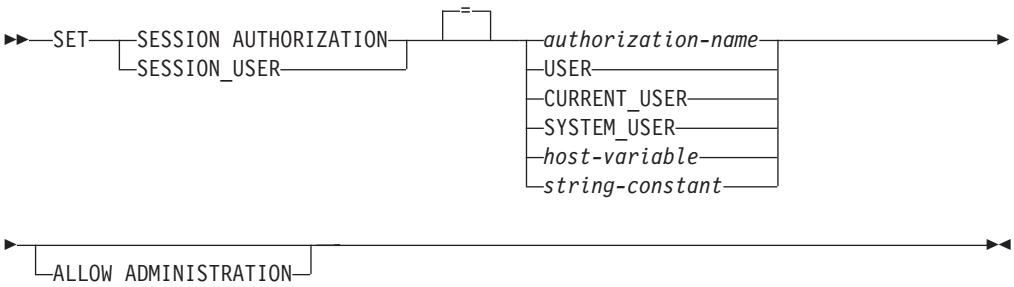
Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include SETSESSIONUSER on the authorization ID value to which the special register is being set.

Syntax



Description

authorization-name
Specifies the authorization ID that is to be used as the new value for the SESSION_USER special register.

USER
The value in the USER special register.

CURRENT_USER
The value in the CURRENT USER special register.

SYSTEM_USER
The value in the SYSTEM_USER special register.

host-variable
A variable of type CHAR or VARCHAR. The length of the contents of *host-variable* must not exceed 128 bytes (SQLSTATE 28000). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 28000).

The characters of *host-variable* must be left-aligned. When specifying *authorization-name* with a host variable, all characters must be specified in uppercase, because there is no conversion to uppercase characters.

string-constant
A character string constant with a maximum length of 128 bytes.

ALLOW ADMINISTRATION
Specifies that SQL schema statements can be specified before this statement in the same unit of work.

Rules

- The value specified for the SESSION_USER special register must conform to the rules for an authorization ID of type USER (SQLSTATE 42602).
- The OWNER bind option specifies the authorization ID that is to be used for static SQL statements.
- This statement can only be issued as the first statement (other than a SET special register statement) in a new unit of work without any open WITH HOLD cursors (SQLSTATE 25001). This restriction includes any PREPARE request for a statement other than a SET special register statement.
- The value of the SESSION_USER special register is used as the authorization ID for all dynamic SQL statements in a package bound with the DYNAMICRULES(RUN) bind option. (This includes INVOKERUN and DEFINERUN when the package is not used by a routine). If a package is using owner, invoker, or definer authorization based on the DYNAMICRULES option, this statement has no effect on dynamic SQL statements issued from within that package.

Notes

- The SET SESSION AUTHORIZATION statement lets you change the session authorization ID. The session authorization ID represents the current user of the connection and is the authorization ID that DB2 considers for all authorization checking relative to dynamic SQL within a DYNAMICRULES run package. The SESSION_USER special register can be used to see the current value of this session authorization ID.
- The initial value of the SESSION_USER special register for a new connection is the same as the value of the SYSTEM_USER special register.
- The group information for the session authorization ID specified in this statement is acquired at the time of statement execution.
- Setting the SESSION_USER special register does not effect either the CURRENT SCHEMA or the CURRENT PATH special register.
- If any error occurs during the setting of the SESSION_USER special register, the register reverts to its previous value.
- This statement should not be used to allow multiple, different users to reuse the same connection, because each user will inherit the ability to change the value of the SESSION_USER special register that the original connection owner had. This statement is dependent upon the value of SYSTEM_USER for privileges checking, and the initial connection authorization ID is not changed by the SET SESSION AUTHORIZATION statement. Moreover, the following behaviors impacting connection reuse are not addressed by this statement:
  - The CONNECT privilege is not checked for the new authorization ID
  - The content of any updatable special register is not reset; in particular, the content of the ENCRYPTION PASSWORD special register is not modified and is available to the new authorization ID for encryption or decryption

  - The content of any declared global temporary table is not affected, and is accessible to the new authorization ID
  - Any existing links to remote servers are not reset
- If the ALLOW ADMINISTRATION clause is specified, the following types of statements or operations can precede the SET SESSION AUTHORIZATION statement:
  - Data definition language (DDL), including the definition of savepoints and the declaration of global temporary tables, but not including SET INTEGRITY
  - GRANT and REVOKE statements
  - LOCK TABLE statement
  - COMMIT and ROLLBACK statements
  - SET of special registers
  - SET of global variables

Examples

- *Example 1:* The following statement sets the SESSION_USER special register.

```
SET SESSION_USER = RAJIV
```

- *Example 2:* Set the session authorization ID (the SESSION_USER special register) to be the value of the system authorization ID, which is the ID that established the connection on which the statement has been issued.

```
SET SESSION AUTHORIZATION SYSTEM_USER
```

SET USAGE LIST STATE

The SET USAGE LIST STATE statement manages the state of a usage list and the associated data and memory.

This statement is not under transaction control.

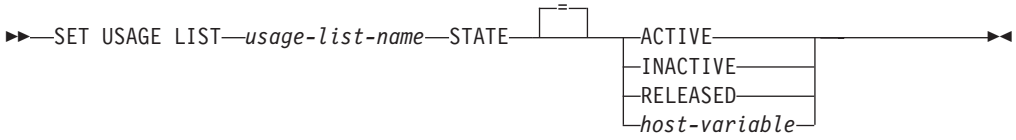
Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include DBADM or SQLADM authority.

Syntax



Description

usage-list-name
Identifies the usage list. The *usage-list-name*, including the implicit or explicit qualifier, must identify a usage list that is described in the catalog (SQLSTATE 42704).

ACTIVE
Indicates that the usage list is activated for monitoring. Memory for the usage list is allocated when the table or index is first referenced by a section. If the usage list is for a partitioned table or index then the memory is allocated when the data partition is first referenced by a section. In a partitioned database environment or DB2 pureScale environment, memory is allocated at each member. If the usage list is already in the ACTIVE state then a warning is returned (SQLSTATE 01598).

On activation, the data in the usage list is removed and collection starts from the beginning of the list.

INACTIVE
Indicates that the usage list is deactivated for monitoring. If the state of a usage list is already set to INACTIVE then this keyword is ignored. If the state of the usage list for a partitioned table or index is set to INACTIVE, then the state of the usage list for each data partition is set to INACTIVE. If the state is already INACTIVE then this keyword is ignored. Similarly, in a partitioned database environment or DB2 pureScale environment, the state of the usage list for each member is set to INACTIVE. If the state is already INACTIVE then this keyword is ignored.

Data collected in the list is not removed when the state of the usage list is set to INACTIVE.

RELEASED
Indicates that the memory associated with a usage list is released. If the state of the usage list for a partitioned table or index is set to RELEASED, then the memory associated with each data partition is released. In a partitioned database environment or DB2 pureScale environment, the memory associated with each member is released.

Notes

- ***Determining current state***: The current state of a usage list is determined by using the MON_GET_USAGE_LIST_STATUS built-in function.
- ***Considerations for DB2 pureScale or partitioned database environments***: If a usage list for a partitioned table or index is activated, memory is allocated for each data partition. Similarly, in a partitioned database environment or DB2 pureScale environment, memory is allocated at each active member.
- ***Memory allocation for unavailable members***: If a member is unavailable at the time of activation, then the memory associated with the usage list for this member is allocated when the member is next activated (if the state of the usage list is still active). This also applies when a member is added to the cluster.
- ***Memory allocation for data partitions that are being added or attached***: For data partitions that are being added or attached, the memory associated with the usage list for this newly added or attached data partition is allocated when the next section that references the partitioned table or index is executed.
- ***Setting INACTIVE independently***: If the usage list was created with the property, WHEN FULL DEACTIVATE, then the state of the usage list for each data partition or member is set to INACTIVE independently.
- ***Implicit reactivation of an active usage list***: If the state of an INACTIVE ON START DATABASE usage list is set to ACTIVE in a partitioned database environment or DB2 pureScale environment, then its behavior is similar to ACTIVE ON START DATABASE until the usage list is explicitly deactivated or the instance is recycled. That is, if state of the usage list is active when a database member is deactivated or offline, and that database member is subsequently reactivated, the usage list for this member is implicitly reactivated.
- ***Definition of released state***: A usage list is considered to be in the released state if it is defined and has not been activated (explicitly or automatically) or has been released using the SET USAGE LIST STATE statement. Usage lists in the state released are not returned by the MON_GET_USAGE_LIST_STATUS table function.
- ***Activation pending, active, and failed states***: If a usage list is activated (explicitly or automatically) then the state of the usage list is set to activation pending and the memory is allocated when the table or index is first referenced by the section. At this point the state of the usage list is set to active. If the memory for the usage list cannot be allocated, then the state of the usage list is set to failed and it must be explicitly activated using the SET USAGE LIST STATE statement.
- ***Inactive usage lists remain inactive upon database member reactivation***: If the state of an ACTIVE ON START DATABASE usage list is set to INACTIVE in a partitioned database environment or DB2 pureScale environment, then its behavior is similar to INACTIVE ON START DATABASE until the usage list is explicitly activated or the instance is recycled. That is, if the state of a usage list is inactive when a database member is deactivated or offline, and that database member is subsequently reactivated, the state of the usage list for this member will remain inactive.

- ***Activating, deactivating, or releasing a usage list for a partitioned table or index***: If a usage list for a partitioned table or index is activated, deactivated, or released then the state change applies to each data partition. Similarly, in a partitioned database environment or DB2 pureScale environment, the state change applies to each member.
- ***Usage list size considerations***: When activated, the memory associated with the usage list is allocated from the monitor heap. At the maximum list size setting, the usage list is approximately 2MB. For partitioned tables or indexes, memory is allocated for each data partition. For example, if a partitioned table has three data partitions defined, the total memory allocated is approximately 6MB. Therefore, activating multiple usage lists imposes more memory requirements on the monitor heap. It is therefore suggested that a reasonable list size is selected or that you set the **mon_heap_sz** configuration parameter to AUTOMATIC so that the database manager manages the monitor heap size.
- ***Data collection when a usage list is set to INACTIVE***: Data collected in the list is not removed when the state of the usage list is set to INACTIVE.
- ***Data access and memory***: The data in the list is still accessible (using MON_GET_TABLE_USAGE_LIST and MON_GET_INDEX_USAGE_LIST table functions) provided that the memory for the list is allocated.
- ***Releasing memory***: The memory associated with the usage list is released when one of the following events occurs:
  - The usage list is dropped.
  - The table or index on which the usage list is defined is dropped. The memory that is associated with the usage is released for all data partitions. In a partitioned database environment or DB2 pureScale environment, the memory that is associated with the usage list is released for all active members.
  - When a data partition is detached from a partitioned table or index. Only the memory associated with the data partition is released.
  - When a database member is deactivated. Only the memory associated with the member is released.
  - When the entire instance or database is deactivated. Usage list data does not persist when the database is deactivated and restarted.
  - When memory associated with the usage list is explicitly released using the SET USAGE LIST STATE statement.

SET variable

The SET variable statement assigns values to variables.

This statement is not under transaction control.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

To reference a transition variable, the privileges held by the authorization ID of the trigger creator must include at least one of the following authorities:

- UPDATE privilege on any columns referenced on the left side of the assignment, and SELECT privilege on any columns referenced on the right side
- CONTROL privilege on the table (subject table of the trigger)
- DATAACCESS authority

If a global variable is referenced in the right side of the assignment statement, the privileges held by the authorization ID of the statement must include one of the following authorities:

- READ privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

If a global variable is assigned a value in the left side of the assignment statement, the privileges held by the authorization ID of the statement must include one of the following authorities:

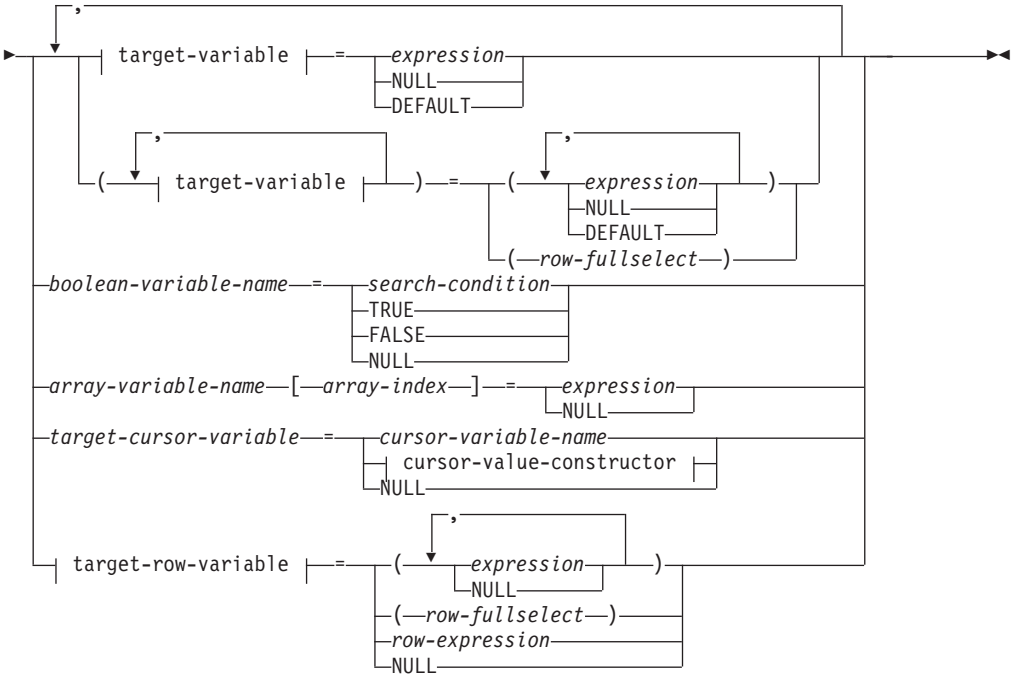
- WRITE privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

To execute this statement with a *row-fullselect* as the right side of the assignment, the privileges held by the authorization ID of the statement must include the privileges necessary to execute the *row-fullselect*. See the Authorization section in "SQL queries".

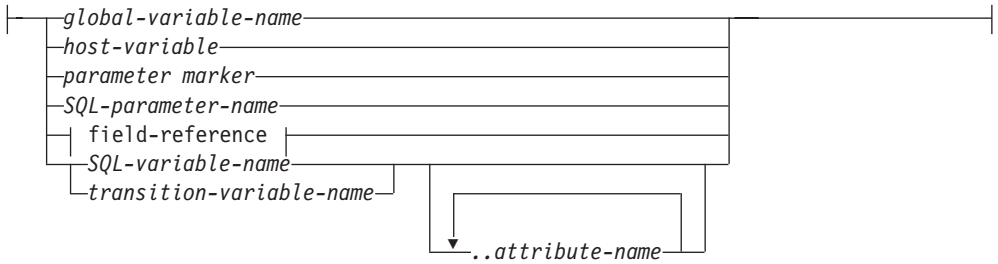
To execute this statement with a *cursor-value-constructor* that uses a *select-statement*, the privileges held by the authorization ID of the statement must include the privileges necessary to execute the *select-statement*. See the Authorization section in "SQL queries".

Syntax

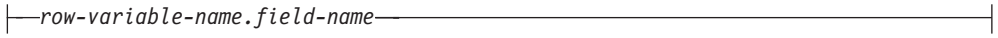
SET variable



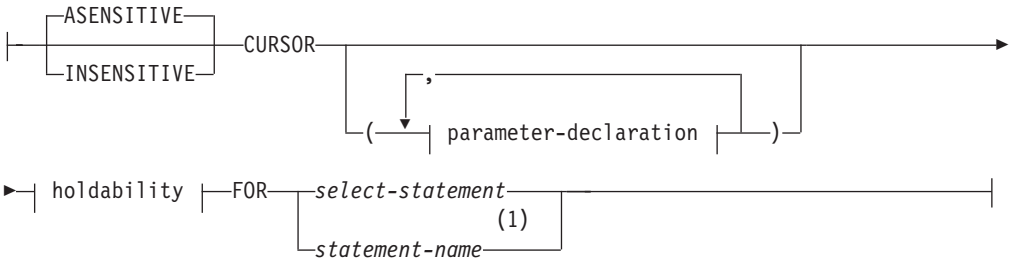
target-variable:



field-reference:

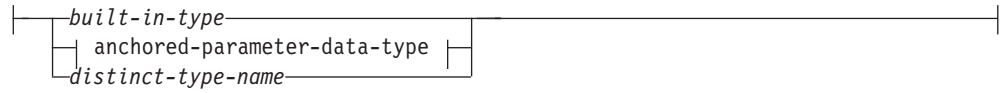


cursor-value-constructor:

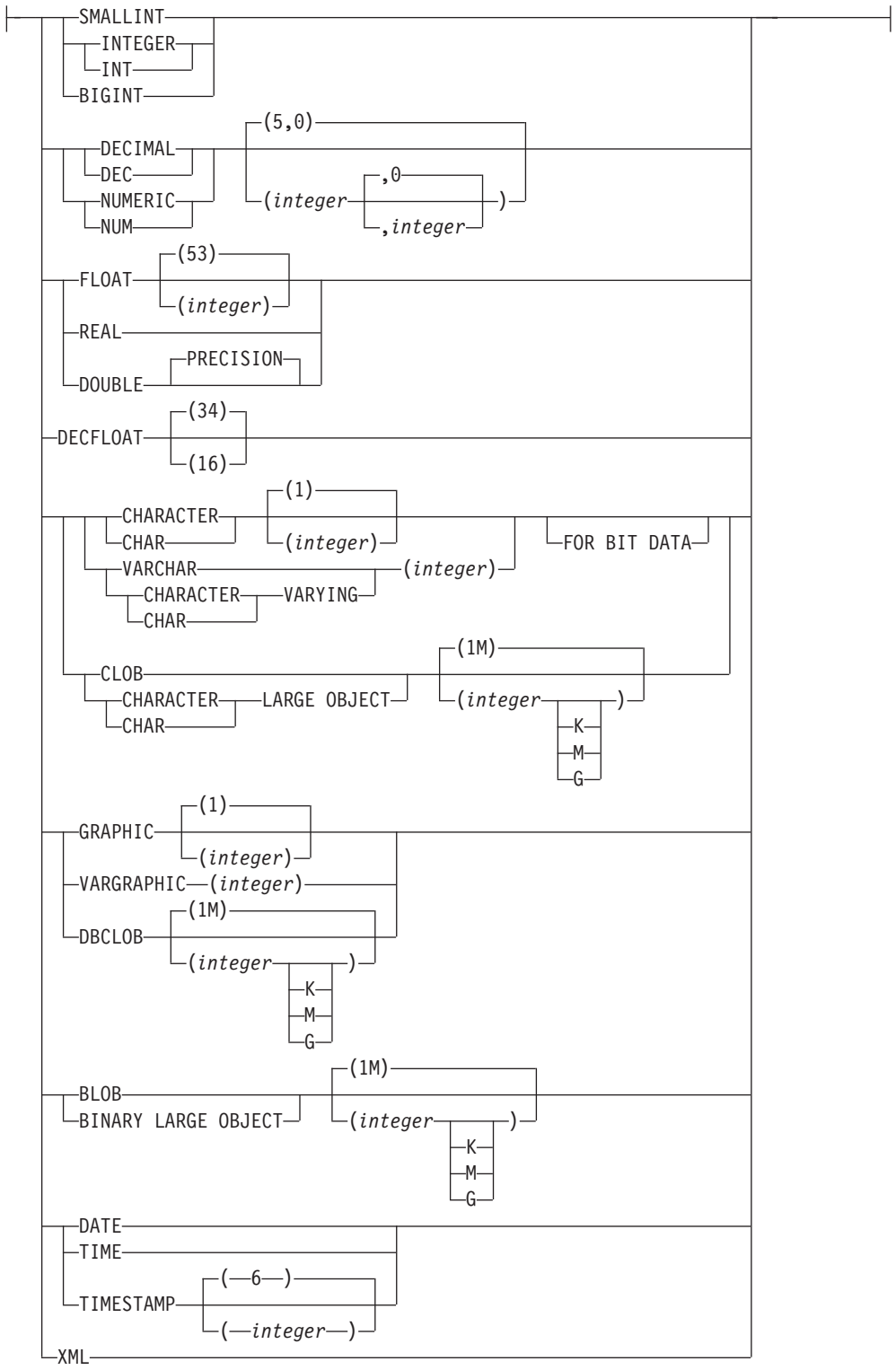


parameter-declaration:

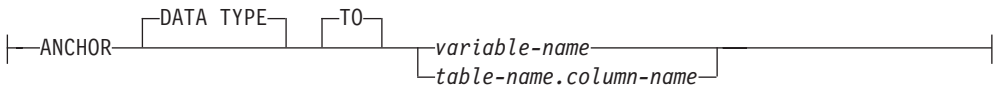
data-type:



built-in-type:

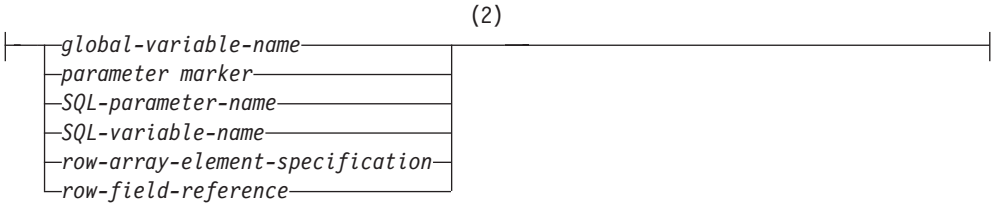


anchored-parameter-data-type:



holdability:

target-row-variable:



Notes:

1 *statement-name* cannot be specified if *parameter-declaration* is specified.

2 The data type must be a row type.

Description

target-variable
Identifies the target variable of the assignment. A *target-variable* representing the same variable must not be specified more than once (SQLSTATE 42701).

global-variable-name
Identifies the global variable that is the assignment target. The *global-variable-name* must identify a global variable that exists at the current server (SQLSTATE 42704).

host-variable
Identifies the host variable that is the assignment target.

parameter-marker
Identifies the parameter marker that is the assignment target.

SQL-parameter-name
Identifies the parameter that is the assignment target. The parameter must be specified in *parameter-declaration* in the CREATE PROCEDURE statement.

field-reference
Identifies the field within a row type value that is the assignment target.

row-variable-name
The name of a variable with a data type that is a row type.

field-name
The name of a field within the row type.

SQL-variable-name
Identifies the SQL variable that is the assignment target. SQL variables must be declared before they are used.

transition-variable-name
Identifies the column to be updated in the transition row. A

transition-variable-name must identify a column in the subject table of a trigger, optionally qualified by a correlation name that identifies the new value (SQLSTATE 42703).

..attribute-name
Specifies the attribute of a structured type that is set (referred to as an *attribute assignment*). The *SQL-variable-name* or *transition-variable-name* specified must be defined with a user-defined structured type (SQLSTATE 428DP). The *..attribute-name* must be an attribute of the structured type (SQLSTATE 42703). An assignment that does not involve the *..attribute-name* clause is referred to as a *conventional assignment*.

expression
Indicates the new value of the target of the assignment. The expression is any expression of the type described in "Expressions". The expression cannot include an aggregate function except when it occurs within a scalar fullselect (SQLSTATE 42903). In the context of a CREATE TRIGGER statement, an *expression* can contain references to OLD and NEW transition variables. The transition variables must be qualified by the *correlation-name* (SQLSTATE 42702).

NULL
Specifies the null value. If the target of the assignment is a row variable, each field is assigned the null value. NULL cannot be the value in an attribute assignment unless it was specifically cast to the data type of the attribute (SQLSTATE 429B9).

DEFAULT
Specifies that the default value should be used.

In SQL procedures, the DEFAULT clause can be specified only for static SQL statements. The exception is that the DEFAULT clause can be specified when *target-variable* is a global variable in a dynamic SQL statement.

If *target-variable* is a column, the value inserted depends on how the column was defined in the table.

- If the column was defined using the WITH DEFAULT clause, the value is set to the default defined for the column (see *default-clause* in "ALTER TABLE").
- If the column was defined using the IDENTITY clause, the value is generated by the database manager.
- If the column was defined without specifying the WITH DEFAULT clause, the IDENTITY clause, or the NOT NULL clause, the value is NULL.
- If the column was defined using the NOT NULL clause and:
  - The IDENTITY clause is not used or
  - The WITH DEFAULT clause was not used or
  - DEFAULT NULL was used

  the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).

If *target-variable* is an SQL variable, the value inserted is the default, as specified or implied in the variable declaration.

If *target-variable* is a global variable, the value inserted is the default, as specified in the variable creation.

If *target-variable* is an SQL variable or an SQL parameter in an SQL procedure, a host variable, or a parameter marker, the DEFAULT keyword cannot be specified (SQLSTATE 42608).

row-fullselect
A fullselect that returns a single row with the number of columns corresponding to the number of target variables or fields in the row variable specified for assignment. The values are assigned to each corresponding target variable or field. If the result of the row fullselect is no rows, null values are assigned to the target variables in the list or, in an assignment to a row variable, a single null is assigned. In the context of a CREATE TRIGGER statement, a *row-fullselect* can contain references to OLD and NEW transition variables, which must be qualified by their *correlation-name* to specify which transition variable is to be used (SQLSTATE 42702). An error is returned if there is more than one row in the result (SQLSTATE 21000).

boolean-variable-name
Identifies an SQL variable or parameter or a global variable. The variable or parameter must be of Boolean type (SQLSTATE 428H0). The SET statement must be issued within a compound SQL (compiled) statement (SQLSTATE 428H2).

search-condition
A search condition whose result is true, false, or unknown. A result of unknown is returned as the Boolean value NULL.

TRUE
Specifies the Boolean value TRUE.

FALSE
Specifies the Boolean value FALSE.

NULL
Specifies the Boolean value NULL.

array-variable-name
Identifies an SQL variable, SQL parameter, or global variable of an array type (SQLSTATE 428H0).

[*array-index***]**
An expression that specifies which element in the array will be the target of the assignment. For an ordinary array, the array-index must be assignable to INTEGER (SQLSTATE 22018 or 428H1). Its value must be between 1 and the maximum cardinality defined for the array and cannot be the null value (SQLSTATE 2202E).

For an associative array, the array index expression must be assignable to the index data type of the associative array (SQLSTATE 22018 or 428H1) and cannot be the null value (SQLSTATE 2202E).

target-cursor-variable
Identifies a cursor variable. The data type of *target-cursor-variable* must be a cursor type (SQLSTATE 42821).

cursor-variable-name
Identifies a cursor variable of the same cursor type as *target-cursor-variable*.

cursor-value-constructor
A *cursor-value-constructor* specifies the *select-statement* that is associated with the target variable. The assignment of a *cursor-value-constructor* to a cursor variable defines the underlying cursor of that cursor variable.

ASENSITIVE or INSENSITIVE
Specifies whether the cursor is asensitive or insensitive to changes. See "DECLARE CURSOR" for more information. The default is ASENSITIVE.

ASENSITIVE
Specifies that the cursor should be as sensitive as possible to inserts, updates, or deletes made to the rows underlying the result table, depending on how the *select-statement* is optimized. ASENSITIVE is the default.

INSENSITIVE
Specifies that the cursor does not have sensitivity to inserts, updates, or deletes that are made to the rows underlying the result table. If INSENSITIVE is specified, the cursor is read-only and the result table is materialized when the cursor is opened. As a result, the size of the result table, the order of the rows, and the values for each row do not change after the cursor is opened. The SELECT statement cannot contain a FOR UPDATE clause, and the cursor cannot be used for positioned updates or deletes.

(parameter-declaration, ...)
Specifies the input parameters of the cursor, including the name and the data type of each parameter. Named input parameters can be specified only if *select-statement* is also specified in *cursor-value-constructor* (SQLSTATE 428HU).

parameter-name
Names the cursor parameter for use as an SQL variable within *select-statement*. The name cannot be the same as any other parameter name for the cursor. Names should also be chosen to avoid any column names that could be used in *select-statement*, since column names are resolved before parameter names.

data-type
Specifies the data type of the cursor parameter used within *select-statement*. Structured types, and reference types cannot be specified (SQLSTATE 429BB).

built-in-type
Specifies a built-in data type. For a more complete description of each built-in data type, see "CREATE TABLE".

anchored-parameter-data-type
Identifies another object used to determine the data type of the cursor parameter. The data type of the anchor object is bound by the same limitations that apply when specifying the data type directly.

ANCHOR DATA TYPE TO
Indicates an anchored data type is used to specify the data type.

variable-name
Identifies a local SQL variable, an SQL parameter, or a global variable. The data type of the referenced variable is used as the data type for the cursor parameter.

table-name.column-name
Identifies a column name of an existing table or view. The data type of the column is used as the data type for the cursor parameter.

distinct-type-name
Specifies the name of a distinct type. If *distinct-type-name* is

specified without a schema name, the distinct type is resolved by searching the schemas in the SQL path.

holdability
Specifies whether the cursor is prevented from being closed as a consequence of a commit operation. See "DECLARE CURSOR" for more information. The default is WITHOUT HOLD.

WITHOUT HOLD
Does not prevent the cursor from being closed as a consequence of a commit operation.

WITH HOLD
Maintains resources across multiple units of work. Prevents the cursor from being closed as a consequence of a commit operation.

select-statement
Specifies the SELECT statement of the cursor. See "select-statement" for more information. If *parameter-declaration* is included in *cursor-value-constructor*, then *select-statement* must not include any local SQL variables or routine SQL parameters (SQLSTATE 42704).

statement-name
Specifies the prepared *select-statement* of the cursor. See "PREPARE" for an explanation of prepared statements. The target cursor variable must not have a data type that is a strongly-typed user-defined cursor type (SQLSTATE 428HU). Named input parameters must not be specified in *cursor-value-constructor* if *statement-name* is specified (SQLSTATE 428HU).

target-row-variable
Identifies the target row variable of the assignment. The data type must be of a row type.

row-expression
Specifies the new row value for the target of the assignment. It can be any row expression of the type described in "Row expression". The number of fields in the row must match the target of the assignment and each field in the row must be assignable to the corresponding field in the target of the assignment. If the source and the target values are a user-defined row type, the type names must be the same (SQLSTATE 42821).

Rules

- The number of values to be assigned from expressions, NULLs, DEFAULTs, or the *row-fullselect* must match the number of *target-variables* specified for assignment (SQLSTATE 42802).
- A SET variable statement cannot assign an SQL variable and a transition variable in one statement (SQLSTATE 42997).
- Global variables cannot be assigned inside triggers that are not defined using a compound SQL (compiled) statement, functions that are not defined using a compound SQL (compiled) statement, methods, or compound SQL (inlined) statements (SQLSTATE 428GX).
- If the value being assigned is an array resulting from an array constructor or from ARRAY_AGG, the base types of the array and of the target variable must be identical (SQLSTATE 42821).
- ***Use of anchored data types:*** An anchored data type cannot refer to (SQLSTATE 428HS): a nickname, typed table, typed view, declared temporary table, row definition associated with a weakly typed cursor, object with a code page or collation that is different from the database code page or database collation.

- ***Assignments involving cursor variables:*** Assignments that reference a cursor variable that set it to the value of a cursor value constructor can only be used in compound SQL (compiled) statements. Any OPEN statement using a cursor variable must occur within the same scope as the assignment (SQLSTATE 51044).

Notes

- Values are assigned to target variables according to specific assignment rules.
- ***Assignment statement in SQL procedures:*** Assignment statements in SQL procedures must conform to the SQL assignment rules. String assignments use retrieval assignment rules.
- ***Assignments of array elements:*** If the assignment is of the form `SET A[idx] = rhs`, where `A` is an array variable name, `idx` is an expression used as the array-index, and `rhs` is an expression of the same type as the array element, then:
  1. If array `A` is the null value, set `A` to the empty array.
  2. Let `C` be the cardinality of array `A`.
  3. If `A` is an ordinary array:
     - If `idx` is less than or equal to `C`, the value in the position identified by `idx` is replaced by the value of `rhs`.
     - If `idx` is greater than `C`, then:
       - The value in position *i*, for *i* greater than `C` and less than `idx`, is set to the null value.
       - The value in position `idx` is set to the value of `rhs`.
       - The cardinality of `A` is set to `idx`.
  4. If `A` is an associative array:
     - If `idx` matches an existing array index value, the element value with array index `idx` is replaced by the value of `rhs`.
     - If `idx` does not match any existing array index value, then:
       - The cardinality of `A` is incremented by 1
       - The new element value is set to `rhs` with associated array index value `idx`.
  5. If `idx` is less than or equal to `C`, the value in the position identified by `idx` is replaced by the value of `rhs`.
  6. If `idx` is greater than `C`, then:
     a. The value in position *i*, for *i* greater than `C` and less than `idx`, is set to the null value.
     b. The value in position `idx` is set to the value of `rhs`.
     c. The cardinality of `A` is set to `idx`.
- If a variable has been declared with an identifier that matches the name of a special register (such as PATH), the variable must be delimited to prevent unintentional assignment to the special register (for example, `SET "PATH" = 1;` for a variable called PATH that has been declared as an integer).
- If more than one assignment is included, each *expression* and *row-fullselect* is evaluated before the assignments are performed. Thus, references to target variables in an expression or row fullselect are always the value of the target variable before any assignment in the single SET statement.
- When an identity column defined as a distinct type is updated, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type before the computation.)

- To have the database manager generate a value on a SET statement for an identity column, use the DEFAULT keyword:

```
SET NEW.EMPNO = DEFAULT
```

  In this example, NEW.EMPNO is defined as an identity column, and the value used to update this column is generated by the database manager.
- For more information about consuming values of a generated sequence for an identity column, and for information about exceeding the maximum value for an identity column, see "INSERT".

Examples

- *Example 1:* Set the salary column of the row for which the trigger action is currently executing to 50000.

```
SET NEW_VAR.SALARY = 50000;
```

  Or:

```
SET (NEW_VAR.SALARY) = (50000);
```

- *Example 2:* Set the salary and the commission column of the row for which the trigger action is currently executing to 50000 and 8000, respectively.

```
SET NEW_VAR.SALARY = 50000, NEW_VAR.COMM = 8000;
```

  Or:

```
SET (NEW_VAR.SALARY, NEW_VAR.COMM) = (50000, 8000);
```

- *Example 3:* Set the salary and the commission column of the row for which the trigger action is currently executing to the average salary and commission of employees in the department that is associated with the updated row.

```
SET (NEW_VAR.SALARY, NEW_VAR.COMM)
   = (SELECT AVG(SALARY), AVG(COMM)
        FROM EMPLOYEE E
        WHERE E.WORKDEPT = NEW_VAR.WORKDEPT);
```

- *Example 4:* Set the salary and the commission column of the row for which the trigger action is currently executing to 10000 and the original value of salary (that is, before the SET statement was executed), respectively.

```
SET NEW_VAR.SALARY = 10000, NEW_VAR.COMM = NEW_VAR.SALARY;
```

  Or:

```
SET (NEW_VAR.SALARY, NEW_VAR.COMM) = (10000, NEW_VAR.SALARY);
```

- *Example 5:* Increase the SQL variable P_SALARY by 10 percent.

```
SET P_SALARY = P_SALARY + (P_SALARY * .10)
```

- *Example 6:* Set the SQL variable P_SALARY to the null value.

```
SET P_SALARY = NULL
```

- *Example 7:* Assign numbers 2.71828183 and 3.1415926 to the first and tenth elements of the array variable SPECIALNUMBERS. After the first assignment, the cardinality of P_PHONENUMBERS is 1. After the second assignment, the cardinality is 10, and elements 2 to 9 have been implicitly assigned the null value.

```
SET SPECIALNUMBERS[1] = 2.71828183;

SET SPECIALNUMBERS[10] = 3.14159265;
```

- *Example 8:* Given a table named SECURITY.USERS, which has a row for every user that could connect to the database, assign the current time and the authorization level to the global variables USERINFO.GV_CONNECT_TIME and USERINFO.GV_AUTH_LEVEL, respectively.

```
SET USERINFO.GV_CONNECT_TIME = CURRENT TIMESTAMP,
  USERINFO.GV_AUTH_LEVEL = (
    SELECT AUTHLEVEL FROM SECURITY.USERS
      WHERE USERID = CURRENT USER)
```

- *Example 9:* Assign values to associative array variable, CAPITALS, which has been declared as the array type CAPITALSARRAY.

```
SET CAPITALS['British Columbia'] = 'Victoria';
SET CAPITALS['Alberta'] = 'Edmonton';
SET CAPITALS['Manitoba'] = 'Winnipeg';
SET CAPITALS['Canada'] = 'Ottawa';
```

  When populating the CAPITALS array, the array indexes are province, territory, and country names specified by strings and the associated array elements are capital cities, also specified by strings.

- *Example 10:* Assign easy to remember names as indexes for personal phone numbers stored in the array variable PHONELIST of array type PERSONAL_PHONENUMBERS.

```
SET PHONELIST['Home'] = '4163053745';
SET PHONELIST['Work'] = '4163053746';
SET PHONELIST['Mom']  = '4164789683';
```

SIGNAL

The SIGNAL statement is used to signal an error or warning condition. It causes an error or warning to be returned with the specified SQLSTATE, along with optional message text.

Invocation

This statement can be embedded in an:

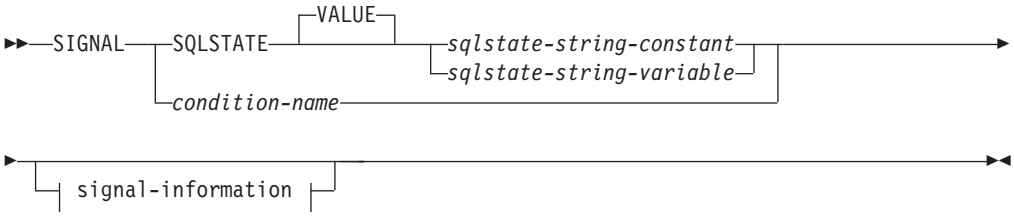
- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

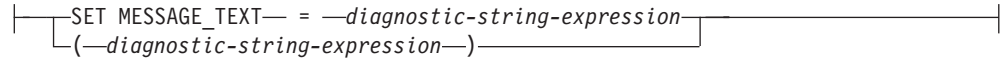
Authorization

If a module condition is referenced, the privileges held by the authorization ID of the statement must include EXECUTE privilege on the module.

Syntax



signal-information:



Description

SQLSTATE VALUE

Specifies the SQLSTATE that will be returned. Any valid SQLSTATE value can be used. The specified value must follow the rules for SQLSTATEs:

- Each character must be from the set of digits ('0' through '9') or upper case letters ('A' through 'Z') without diacritical marks
- The SQLSTATE class (first two characters) cannot be '00', since this represents successful completion.

In the context of either a compound SQL (inlined) statement, the following rules must also be applied:

- The SQLSTATE class (first two characters) cannot be '01' or '02', since these are not error classes.

- If the SQLSTATE class starts with the numbers '0' through '6' or the letters 'A' through 'H', then the subclass (the last three characters) must start with a letter in the range of 'I' through 'Z'.
- If the SQLSTATE class starts with the numbers '7', '8', '9', or the letters 'I' through 'Z', then the subclass can be any of '0' through '9' or 'A' through 'Z'.

If the SQLSTATE does not conform to these rules, an error is returned.

sqlstate-string-constant
The *sqlstate-string-constant* must be a character string constant with exactly 5 characters.

sqlstate-string-variable
The specified SQL variable or SQL parameter must be of data type CHAR(5) and must not be the null value.

condition-name
Specifies the name of a condition that will be returned. The condition-name must be declared within the compound-statement or identify a condition that exists at the current server (SQLSTATE 42373).

SET MESSAGE_TEXT =
Specifies a string that describes the error or warning. The string is returned in the SQLERRMC field of the SQLCA. If the actual string is longer than 70 bytes, it is truncated without warning.

diagnostic-string-expression
A literal string, or a local variable or parameter that describes the error condition. If the string is longer than 70 bytes, it is truncated.

(*diagnostic-string-expression***)**
An expression of type CHAR or VARCHAR that returns a character string of up to 70 bytes to describe the error condition. If the string is longer than 70 bytes, it is truncated. This option is only provided within the scope of a CREATE TRIGGER statement for compatibility with previous versions of DB2. Regular use is not recommended.

Notes

- If a SIGNAL statement is issued using a *condition-name* that has no associated SQLSTATE value and the condition is not handled, SQLSTATE 45000 is returned and the SQLCODE is set to -438. Note that such a condition will not be handled by a condition handler for SQLSTATE 45000 that is within the scope of the routine issuing the SIGNAL statement.
- If a SIGNAL statement is issued using an SQLSTATE value or a *condition-name* with an associated SQLSTATE value, the SQLCODE returned is based on the SQLSTATE value as follows:
  - If the specified SQLSTATE class is either '01' or '02', a warning or not found condition is returned and the SQLCODE is set to +438.
  - Otherwise, an exception condition is returned and the SQLCODE is set to -438.
- A SIGNAL statement has the indicated fields of the SQLCA set as follows:
  - `sqlerrd` fields are set to zero
  - `sqlwarn` fields are set to blank
  - `sqlerrmc` is set to the first 70 bytes of MESSAGE_TEXT
  - `sqlerrml` is set to the length of `sqlerrmc`, or to zero if no SET MESSAGE_TEXT clause is specified
  - `sqlerrp` is set to ROUTINE

- SQLSTATE values are composed of a two-character class code value, followed by a three-character subclass code value. Class code values represent classes of successful and unsuccessful execution conditions.

  Any valid SQLSTATE value can be used in the SIGNAL statement. However, it is recommended that programmers define new SQLSTATEs based on ranges reserved for applications. This prevents the unintentional use of an SQLSTATE value that might be defined by the database manager in a future release.

  – SQLSTATE classes that begin with the characters '7' through '9', or 'I' through 'Z' may be defined. Within these classes, any subclass may be defined.
  – SQLSTATE classes that begin with the characters '0' through '6', or 'A' through 'H' are reserved for the database manager. Within these classes, subclasses that begin with the characters '0' through 'H' are reserved for the database manager. Subclasses that begin with the characters 'I' through 'Z' may be defined.

Example

An SQL procedure for an order system that signals an application error when a customer number is not known to the application. The ORDERS table includes a foreign key to the CUSTOMER table, requiring that the CUSTNO exist before an order can be inserted.

```
CREATE PROCEDURE SUBMIT_ORDER
  (IN ONUM INTEGER, IN CNUM INTEGER,
   IN PNUM INTEGER, IN QNUM INTEGER)
   SPECIFIC SUBMIT_ORDER
   MODIFIES SQL DATA
  LANGUAGE SQL
  BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE VALUE '23503'
      SIGNAL SQLSTATE '75002'
        SET MESSAGE_TEXT = 'Customer number is not known';
    INSERT INTO ORDERS (ORDERNO, CUSTNO, PARTNO, QUANTITY)
      VALUES (ONUM, CNUM, PNUM, QNUM);
  END
```

TRANSFER OWNERSHIP

The TRANSFER OWNERSHIP statement transfers ownership of a database object.

Invocation

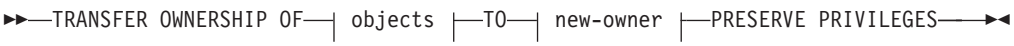
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

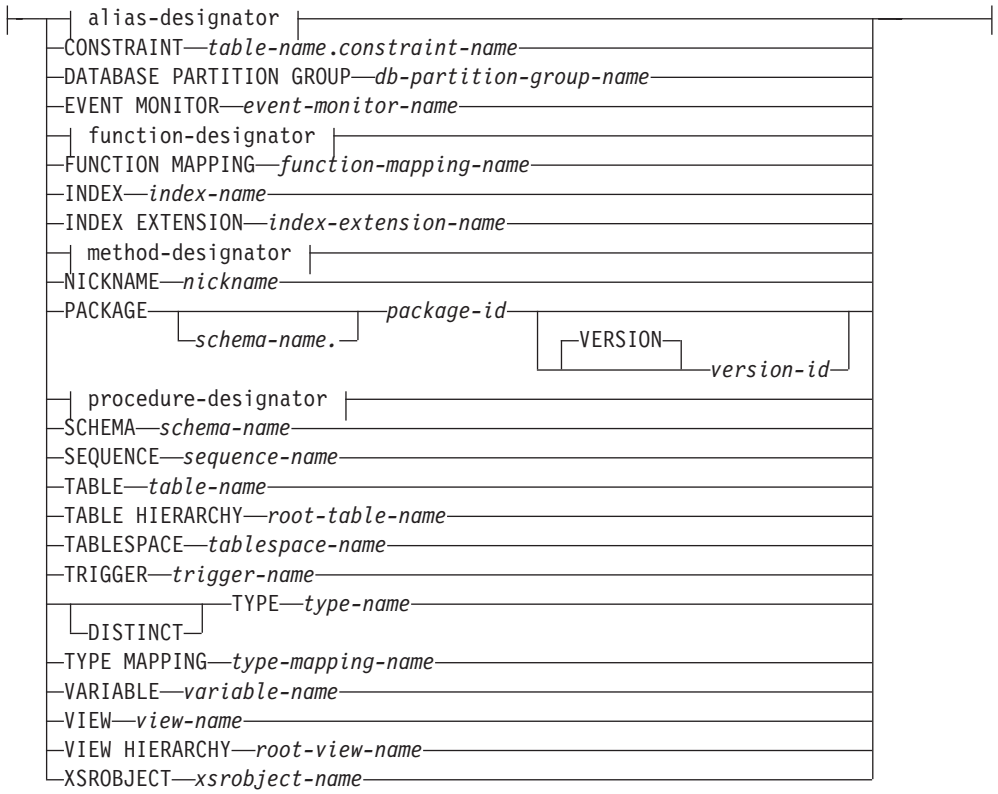
The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- Ownership of the object
- SECADM authority

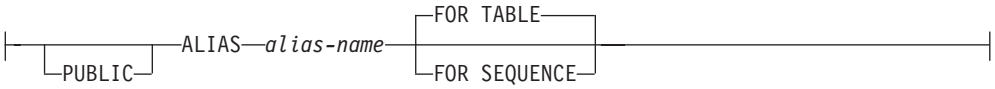
Syntax



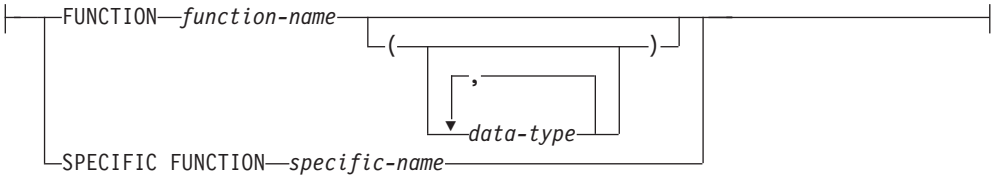
objects:



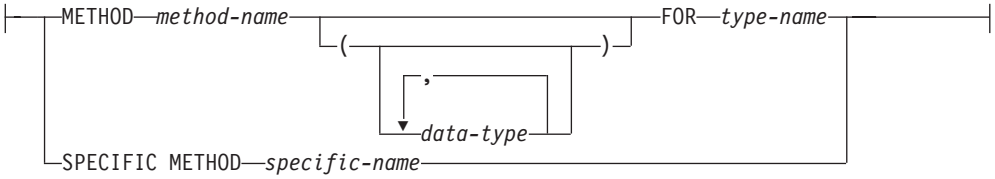
alias-designator:



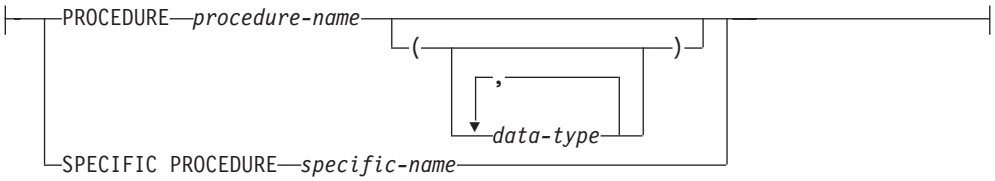
function-designator:



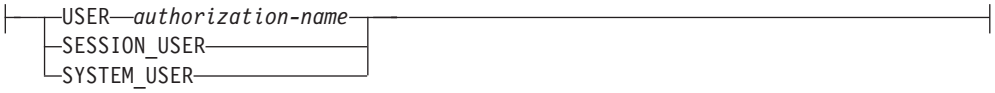
method-designator:



procedure-designator:



new-owner:



Description

alias-designator

ALIAS *alias-name*
Identifies the alias that is to have its ownership transferred. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). If PUBLIC is specified, the *alias-name* must identify a public alias that exists at the current server (SQLSTATE 42704).

FOR TABLE, or FOR SEQUENCE
Specifies the object type for the alias.

FOR TABLE
The alias is for a table, view, or nickname. When ownership of the alias is transferred, the value in the OWNER column for the alias in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.

FOR SEQUENCE
The alias is for a sequence. When ownership of the alias is transferred, the value in the OWNER column for the alias in the SYSCAT.SEQUENCES catalog view is replaced with the authorization ID of the new owner.

CONSTRAINT *table-name.constraint-name*
Identifies the constraint that is to have its ownership transferred. The *table-name.constraint-name* combination must identify a constraint and the table that it constrains. The *constraint-name* must identify a constraint that is described in the catalog (SQLSTATE 42704).

When ownership of the constraint is transferred, the value in the OWNER column for the constraint in the SYSCAT.TABCONST catalog view is replaced with the authorization ID of the new owner.

- If the constraint is a FOREIGN KEY constraint, the OWNER column in the SYSCAT.REFERENCES catalog view is replaced with the authorization ID of the new owner.
- If the constraint is a PRIMARY KEY or UNIQUE constraint, the OWNER column in the SYSCAT.INDEXES catalog view for the index that was created implicitly for this constraint is replaced with the authorization ID of the new owner. If the index existed, and it is reused in this case, the owner of the index is not changed.

DATABASE PARTITION GROUP *db-partition-group-name*
Identifies the database partition group that is to have its ownership transferred. The *db-partition-group-name* must identify a database partition group that is described in the catalog (SQLSTATE 42704).

When ownership of the database partition group is transferred, the value in the OWNER column for the database partition group in the SYSCAT.DBPARTITIONGROUPS catalog view is replaced with the authorization ID of the new owner.

EVENT MONITOR *event-monitor-name*
Identifies the event monitor that is to have its ownership transferred. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

When ownership of the event monitor is transferred, the value in the OWNER column for the event monitor in the SYSCAT.EVENTMONITORS catalog view is replaced with the authorization ID of the new owner.

If the identified event monitor is active, an error is returned (SQLSTATE 429BT).

If there are event files in the target path of a WRITE TO FILE event monitor whose ownership is being transferred, the event files are not deleted.

When ownership of WRITE TO TABLE event monitors is transferred, table information in the SYSCAT.EVENTTABLES catalog view is retained.

function-designator
Identifies the function that is to have its ownership transferred. For more information, see “Function, method, and procedure designators” on page 20. The specified function instance must be a user-defined function or function template that is described in the catalog. Ownership of functions that are implicitly generated by CREATE TYPE statements cannot be transferred (SQLSTATE 429BT).

When ownership of the function is transferred, the value in the OWNER column for the function in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner. Transferring ownership of an SQL function that has an associated package also implicitly transfers ownership of the package to the new owner.

SPECIFIC FUNCTION *specific-name*
Identifies the particular user-defined function that is to have its ownership transferred, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific function is transferred, the value in the OWNER column for the specific function in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

FUNCTION MAPPING *function-mapping-name*
Identifies the function mapping that is to have its ownership transferred. The *function-mapping-name* must identify a function mapping that is described in the catalog (SQLSTATE 42704).

When ownership of the function mapping is transferred, the value in the OWNER column for the function mapping in the SYSCAT.FUNCMAPPINGS catalog view is replaced with the authorization ID of the new owner.

INDEX *index-name*
Identifies the index or index specification that is to have its ownership transferred. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704).

When ownership of the index is transferred, the value in the OWNER column for the index in the SYSCAT.INDEXES catalog view is replaced with the authorization ID of the new owner.

Ownership of an index cannot be transferred if the table on which the index is defined is a global temporary table (SQLSTATE 429BT).

INDEX EXTENSION *index-extension-name*
Identifies the index extension that is to have its ownership transferred. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE 42704).

When ownership of the index extension is transferred, the value in the OWNER column for the index extension in the SYSCAT.INDEXEXTENSIONS catalog view is replaced with the authorization ID of the new owner.

method-designator
Identifies the method that is to have its ownership transferred. For more information, see "Function, method, and procedure designators" on page 20. The method body specified must be a method that is described in the catalog (SQLSTATE 42704). The ownership of methods that are implicitly generated by the CREATE TYPE statement cannot be transferred (SQLSTATE 429BT).

When ownership of the method is transferred, the value in the OWNER column for the method in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

NICKNAME *nickname*
Identifies the nickname that is to have its ownership transferred. The *nickname* must be a nickname that is described in the catalog (SQLSTATE 42704).

When ownership of the nickname is transferred, the value in the OWNER column for the nickname in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.

PACKAGE *schema-name.package-id*
Identifies the package that is to have its ownership transferred. If a schema name is not specified, the package identifier is implicitly qualified by the default schema. The schema name and package identifier, together with the implicitly or explicitly specified version identifier, must identify a package that is described in the catalog (SQLSTATE 42704).

VERSION *version-id*
Identifies which package version is to have its ownership transferred. If a value is not specified, the version defaults to the empty string, and the ownership of this package is transferred. If multiple packages with the same package name but different versions exist, only the ownership of the package whose *version-id* is specified in the TRANSFER OWNERSHIP statement is transferred. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

When ownership of the package is transferred, the value in the BOUNDBY column for the package in the SYSCAT.PACKAGES catalog view is replaced with the authorization ID of the new owner.

The ownership of packages that are associated with SQL procedures cannot be transferred (SQLSTATE 429BT).

procedure-designator
Identifies the procedure that is to have its ownership transferred. For more information, see "Function, method, and procedure designators" on page 20. The procedure instance specified must be a procedure that is described in the catalog.

When ownership of the procedure is transferred, the value in the OWNER column for the procedure in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

Transferring ownership of an SQL procedure that has an associated package also implicitly transfers ownership of the package to the new owner.

SPECIFIC PROCEDURE *specific-name*
Identifies the particular procedure that is to have its ownership transferred, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific procedure is transferred, the value in the OWNER column for the specific procedure in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

SCHEMA *schema-name*
Identifies the schema that is to have its ownership transferred. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704).

When ownership of the schema is transferred, the value in the OWNER column and the DEFINER column for the schema in the SYSCAT.SCHEMATA catalog view is replaced with the authorization ID of the new owner.

Ownership of built-in schemas (where the definer is SYSIBM) cannot be transferred (SQLSTATE 42832).

SEQUENCE *sequence-name*
Identifies the sequence that is to have its ownership transferred. The *sequence-name* must identify a sequence that is described in the catalog (SQLSTATE 42704).

When ownership of the sequence is transferred, the value in the OWNER column for the schema in the SYSCAT.SEQUENCES catalog view is replaced with the authorization ID of the new owner.

TABLE *table-name*
Identifies the table that is to have its ownership transferred. The *table-name* must identify a table that exists in the database (SQLSTATE 42704) and must not identify a declared temporary table (SQLSTATE 42995).

When ownership of the table is transferred:

- The value in the OWNER column for the table in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the table in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

Ownership of subtables in a table hierarchy cannot be transferred (SQLSTATE 429BT).

In a federated system, ownership of a remote table that was created using transparent DDL can be transferred. Transferring the ownership of a remote table will not transfer ownership of the nickname that is associated with the table. Ownership of such a nickname can be transferred explicitly using the TRANSFER OWNERSHIP statement.

TABLE HIERARCHY *root-table-name*
Identifies the typed table that is the root table in a typed table hierarchy that is to have its ownership transferred. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR), and must refer to a typed table that exists in the database (SQLSTATE 42704).

When ownership of the table hierarchy is transferred:

- The value in the OWNER column for the root table and all of its subtables in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the table and all of its subtables in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

TABLESPACE *tablespace-name*
Identifies the table space that is to have its ownership transferred. The *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704).

When ownership of the table space is transferred, the value in the OWNER column for the table space in the SYSCAT.TABLESPACES catalog view is replaced with the authorization ID of the new owner.

TRIGGER *trigger-name*
Identifies the trigger that is to have its ownership transferred. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704).

When ownership of the trigger is transferred, the value in the OWNER column for the trigger in the SYSCAT.TRIGGERS catalog view is replaced with the authorization ID of the new owner.

TYPE *type-name*
Identifies the user-defined type that is to have its ownership transferred. The *type-name* must identify a type that is described in the catalog (SQLSTATE 42704). If DISTINCT is specified, *type-name* must identify a distinct type that is described in the catalog (SQLSTATE 42704).

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

When ownership of the type is transferred, the value in the OWNER column for the type in the SYSCAT.DATATYPES catalog view is replaced with the authorization ID of the new owner.

TYPE MAPPING *type-mapping-name*
Identifies the user-defined data type mapping that is to have its ownership transferred. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704).

When ownership of the type mapping is transferred, the value in the OWNER column for the type mapping in the SYSCAT.TYPEMAPPINGS catalog view is replaced with the authorization ID of the new owner.

VARIABLE *variable-name*
Indicates that the object whose ownership is to be transferred is a created global variable. The *variable-name* must identify a global variable that exists at the current server (SQLSTATE 42704).

When the global variable is transferred, the value in the OWNER column for the global variable in the SYSCAT.VARIABLES catalog view is replaced with the authorization ID of the new owner.

VIEW *view-name*
Identifies the view that is to have its ownership transferred. The *view-name* must identify a view that exists in the database (SQLSTATE 42704).

When ownership of the view is transferred:

- The value in the OWNER column for the view in the SYSCAT.VIEWS catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the view in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

The ownership of a subview in a view hierarchy cannot be transferred (SQLSTATE 429BT).

VIEW HIERARCHY *root-view-name*
Identifies the typed view that is the root view in a typed view hierarchy that is to have its ownership transferred. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR), and must refer to a typed view that exists in the database (SQLSTATE 42704).

When ownership of the view hierarchy is transferred:

- The value in the OWNER column for the root view and all of its subviews in the SYSCAT.VIEWS catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the view and all of its subviews in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

XSROBJECT *xsrobject-name*
Identifies the XSR object that is to have its ownership transferred. The *xsrobject-name* must identify an XSR object that is described in the catalog (SQLSTATE 42704).

When ownership of the XSR object is transferred, the value in the OWNER column for the XSR object in the SYSCAT.XSROBJECTS catalog view is replaced with the authorization ID of the new owner.

USER *authorization-name*
Specifies the authorization ID to which ownership of the object is being transferred.

SESSION_USER
Specifies that the value of the SESSION_USER special register is to be used as the authorization ID to which ownership of the object is being transferred.

SYSTEM_USER
Specifies that the value of the SYSTEM_USER special register is to be used as the authorization ID to which ownership of the object is being transferred.

PRESERVE PRIVILEGES
Specifies that the current owner of an object that is to have its ownership transferred will continue to hold any existing privileges on the object after the transfer. For example, any privileges that were granted to the creator of a view when that view was created continue to be held by the original owner even after ownership has been transferred to another user.

Rules

- Ownership of most built-in objects (where the owner is SYSIBM) cannot be transferred (SQLSTATE 42832). However, you can transfer ownership of implicitly created schema objects that have SYSIBM in the OWNER column and do not have SYSIBM in the DEFINER column.
- Ownership of schemas whose name starts with 'SYS' cannot be transferred (SQLSTATE 42832).
- Ownership of the following objects cannot be explicitly transferred (SQLSTATE 429BT):
  - Subtables in a table hierarchy (they are transferred with the root hierarchy table)
  - Subviews in a view hierarchy (they are transferred with the root hierarchy view)

  - Indexes that are defined on global temporary tables
  - Methods or functions that are implicitly generated when a user-defined type is created
  - Module aliases and modules
  - Packages that depend on SQL procedures (they are transferred with the SQL procedure)
  - Event monitors that are active (they can be transferred when they are not active)
- An authorization ID that has SECADM authority cannot transfer the ownership of an object to itself, if it is not already the owner of the object (SQLSTATE 42502).

Notes

- All privileges that the current owner has that were granted as part of the creation of the object are transferred to the new owner. If the current owner has had a privilege on the object revoked, and that privilege was subsequently granted back, the privilege is not transferred. For implicitly created schema objects that have not already been transferred, the new owner is granted CREATEIN, DROPIN, and ALTERIN privileges on the schema and can also grant these privileges to other users.
- When the ownership of a database object is transferred, the new owner must have the set of privileges on the base objects, as indicated by the object's dependencies, that are required to maintain the object's existence unchanged. The new owner does not need the privileges required to create the object if those privileges are not required to maintain the object's existence.

  For example:
  - Consider a view with SELECT and INSERT dependencies on an underlying table. The privileges held by the new owner of the view must include at least SELECT (with or without the GRANT OPTION) and INSERT (with or without the GRANT OPTION) for the ownership transfer to be successful. If the dependencies were SELECT WITH GRANT OPTION and INSERT WITH GRANT OPTION, the privileges held by the new owner of the view must include at least SELECT WITH GRANT OPTION and INSERT WITH GRANT OPTION.
  - Consider a view with a dependency on a routine. The privileges held by the new owner of the view must include at least EXECUTE on the dependent routine.
  - Consider a trigger with a dependency on a table. The privileges held by the new owner of the trigger must include the same set of privileges on the table that are indicated by the trigger's dependencies. ALTER privilege on the table on which the trigger is defined is not required.

  The following table lists the catalog views that describe the objects on which other database objects depend.

Table 36. Catalog Views that Describe Objects on which Other Objects Depend

Database Object	Catalog View
CONSTRAINT	SYSCAT.CONSTDEP
FUNCTION	SYSCAT.ROUTINEDEP; SYSCAT.ROUTINES (for a sourced function)
INDEX	SYSCAT.INDEXDEP
INDEX EXTENSION	SYSCAT.INDEXEXTENSIONDEP

Table 36. Catalog Views that Describe Objects on which Other Objects Depend (continued)

Database Object	Catalog View
METHOD	SYSCAT.ROUTINEDEP
PACKAGE	SYSCAT.PACKAGEDEP
PROCEDURE	SYSCAT.ROUTINEDEP
TABLE	SYSCAT.TABDEP
TRIGGER	SYSCAT.TRIGDEP
VIEW	SYSCAT.TABDEP
XSROBJECT	SYSCAT.XSROBJECTDEP

If ownership of a database object that depends on another object is to be transferred successfully, the new owner of the database object must hold certain privileges on the dependent object of that dependency:

- If the dependent object is a sequence, the new owner must have the USAGE privilege on that sequence.
- If the dependent object is a function, method, or procedure, the new owner must have the EXECUTE privilege on that function, method, or procedure.
- If the dependent object is a package, the new owner must have the EXECUTE privilege on that package.
- If the dependent object is an XSR object, the new owner must have the USAGE privilege on that XSR object.

For any other dependent object of a dependency, use the TABAUTH column in the appropriate catalog view to determine what privileges the new owner must hold.

- If an attempt is made to transfer ownership of an object to its owner, a warning is returned (SQLSTATE 01676).
- Ownership of the following database objects cannot be transferred, because these objects have no owner: audit policies, buffer pools, roles, security labels, security label components, security policies, servers, transformation functions, trusted contexts, user mappings, and wrappers. Note that there is no OWNER column in the SYSCAT.AUDITPOLICIES, SYSCAT.BUFFERPOOLS, SYSCAT.CONTEXTS, SYSCAT.ROLES, SYSCAT.SECURITYLABELS, SYSCAT.SECURITYLABELCOMPONENTS, SYSCAT.SECURITYPOLICIES, SYSCAT.SERVERS, SYSCAT.TRANSFORMS, SYSCAT.USEROPTIONS, and SYSCAT.WRAPPERS catalog views.
- The schema name of an object whose ownership was transferred does not automatically change.
- ***Syntax alternatives***: For consistency with other SQL statements:
  - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
  - SYNONYM can be specified in place of ALIAS

Examples

- *Example 1:* Transfer ownership of table T1 to PAUL.

```
TRANSFER OWNERSHIP OF TABLE WALID.T1
  TO USER PAUL PRESERVE PRIVILEGES
```

The value in the OWNER column for the table WALID.T1 in the SYSCAT.TABLES catalog view is replaced with 'PAUL'. Paul is implicitly granted the following privileges on table WALID.T1 (assuming that the previous owner

of the table did not lose any privileges on it): CONTROL and ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, REFERENCE (WITH GRANT OPTION).

- *Example 2:* Assume that JOHN creates tables T1 and T2, and that MIKE holds SELECT privilege on tables JOHN.T1 and JOHN.T2. MIKE creates view V1 that depends on tables JOHN.T1 and JOHN.T2. Transfer ownership of view V1 to HENRY, who has DBADM authority.

  ```
  TRANSFER OWNERSHIP OF VIEW V1
    TO USER HENRY PRESERVE PRIVILEGES
  ```

  The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'HENRY'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'HENRY', and TABNAME = 'V1'.

- *Example 3:* Assume that HENRY, who holds DBADM authority, creates a trigger TR1 that depends on table T1. Transfer ownership of trigger TR1 to WALID, who does not hold DBADM authority.

  ```
  TRANSFER OWNERSHIP OF TRIGGER TR1
    TO USER WALID PRESERVE PRIVILEGES
  ```

  Ownership of the trigger is transferred successfully, even though Walid does not hold DBADM authority.

- *Example 4:* Assume that JOHN creates tables T1 and T2, and that MIKE holds SELECT privilege on table JOHN.T1 and CONTROL privilege on table JOHN.T2. PAUL holds SELECT privilege on tables JOHN.T1 and JOHN.T2. MIKE creates view V1 that depends on tables JOHN.T1 and JOHN.T2. The view has an entry for the SELECT privilege in SYSCAT.TABAUTH and two SELECT dependencies in SYSCAT.TABDEP for tables JOHN.T1 and JOHN.T2. Transfer ownership of view V1 to PAUL, who is a regular user.

  ```
  TRANSFER OWNERSHIP OF VIEW V1
    TO USER PAUL PRESERVE PRIVILEGES
  ```

  Ownership of the view is transferred successfully, even though Paul does not hold CONTROL privilege on table JOHN.T2. Paul only needs SELECT privilege on tables JOHN.T1 and JOHN.T2 to maintain the view's existence. (The view only has SELECT privilege because Paul did not hold CONTROL privilege on both tables when the view was created and, as a result, he was not granted CONTROL on the view.) The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'PAUL'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog view is replaced with 'PAUL'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'PAUL', and TABNAME = 'V1'.

- *Example 5:* Assume that JOHN creates table T1, and that PUBLIC holds SELECT privilege on JOHN.T1. PAUL holds SELECT privilege on JOHN.T1 explicitly, and creates view V1 that depends on table JOHN.T1. Transfer ownership of view V1 to MIKE, who is not a DBADM, but who holds the required privileges to acquire view ownership through the special group PUBLIC.

  ```
  TRANSFER OWNERSHIP OF VIEW V1
    TO USER MIKE PRESERVE PRIVILEGES
  ```

  Ownership of the view is transferred successfully, because Mike holds SELECT privilege on table JOHN.T1 through PUBLIC. The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'MIKE'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog

view is replaced with 'MIKE'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'MIKE', and TABNAME = 'V1'.

- *Example 6:* Similar to example 5, assume that JOHN creates table T1, and that role R1 holds SELECT privilege on JOHN.T1. PAUL holds SELECT privilege on JOHN.T1 explicitly, and creates view V1 that depends on table JOHN.T1. Transfer ownership of view V1 to MIKE, who is not a DBADM, but who holds the required privileges through membership in role R1 to acquire view ownership.

  ```
  TRANSFER OWNERSHIP OF VIEW V1
    TO USER MIKE PRESERVE PRIVILEGES
  ```

  Ownership of the view is transferred successfully, because Mike holds SELECT privilege on table JOHN.T1 through membership in role R1. The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'MIKE'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog view is replaced with 'MIKE'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'MIKE', and TABNAME = 'V1'.

TRUNCATE

The TRUNCATE statement deletes all of the rows from a table.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities for the table, and all subtables of a table hierarchy:

- DELETE privilege on the table to be truncated
- CONTROL privilege on the table to be truncated
- DATAACCESS authority

To ignore any DELETE triggers that are defined on the table, the privileges held by the authorization ID of the statement must include at least one of the following authorities for the table, and all subtables of a table hierarchy:

- ALTER privilege on the table
- CONTROL privilege on the table
- DBADM authority

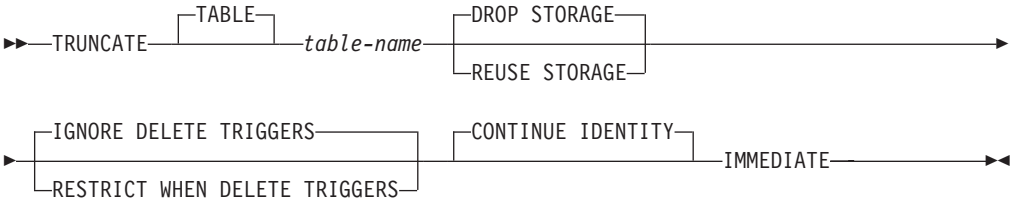
To truncate a table that is protected by a security policy, the privileges held by the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the table
- DBADM authority

To truncate a table that has row access control activated, the authorization ID of the statement must include at least one of the following authorities:

- CONTROL privilege on the table
- DBADM authority

Syntax



Description

table-name
Identifies the table to be truncated. The name must identify a table that exists at the current server (SQLSTATE 42704), but it cannot be a catalog table (SQLSTATE 42832), a nickname (SQLSTATE 42809), a view, a subtable, a

staging table, a system-maintained materialized query table, a system-period temporal table (SQLSTATE 428HZ), or a range-clustered table (SQLSTATE 42807).

If *table-name* is the root table of a table hierarchy, all tables in the table hierarchy will be truncated.

DROP STORAGE or REUSE STORAGE
Specifies whether to drop or reuse the existing storage that is allocated for the table. The default is DROP STORAGE.

DROP STORAGE
All storage allocated for the table is released and made available. If this option is specified (implicitly or explicitly), an online backup would be blocked.

REUSE STORAGE
All storage allocated for the table will continue to be allocated for the table, but the storage will be considered empty. This option is only applicable to tables in DMS table spaces and is ignored otherwise.

IGNORE DELETE TRIGGERS or RESTRICT WHEN DELETE TRIGGERS
Specifies what to do when delete triggers are defined on the table. The default is IGNORE DELETE TRIGGERS.

IGNORE DELETE TRIGGERS
Any delete triggers that are defined for the table are not activated by the truncation operation.

RESTRICT WHEN DELETE TRIGGERS
An error is returned if delete triggers are defined on the table (SQLSTATE 428GJ).

CONTINUE IDENTITY
If an identity column exists for the table, the next identity column value generated continues with the next value that would have been generated if the TRUNCATE statement had not been executed.

IMMEDIATE
Specifies that the truncate operation is processed immediately and cannot be undone. The statement must be the first statement in a transaction (SQLSTATE 25001).

The truncated table is immediately available for use in the same unit of work. Although a ROLLBACK statement is allowed to execute after a TRUNCATE statement, the truncate operation is not undone, and the table remains in a truncated state. For example, if another data change operation is done on the table after the TRUNCATE IMMEDIATE statement and then the ROLLBACK statement is executed, the truncate operation will not be undone, but all other data change operations are undone.

Rules

- ***Referential Integrity:*** The table, and all tables in a table hierarchy, must not be a parent table in an enforced referential constraint (SQLSTATE 428GJ). A self-referencing RI constraint is permitted.
- ***Partitioned tables:*** The table must not be in set integrity pending state due to being altered to attach a data partition (SQLSTATE 55019). The table needs to be checked for integrity before executing the TRUNCATE statement. With DB2 Version 9.7 Fix Pack 1 and later releases, the table must not have any logically

detached partitions (SQLSTATE 55057). The asynchronous partition detach task must complete before executing the TRUNCATE statement.

- ***Exclusive Access:*** No other session can have a cursor open on the table, or a lock held on the table (SQLSTATE 25001).
- ***WITH HOLD cursors:*** The current session cannot have a WITH HOLD cursor open on the table (SQLSTATE 25001).

Notes

- ***Table statistics:*** The statistics for the table are not changed by the TRUNCATE statement.
- ***Number of rows deleted:*** SQLERRD(3) in the SQLCA is set to -1 for the truncate operation. The number of rows that were deleted from the table is not returned.

Examples

- *Example 1:* Empty an unused inventory table regardless of any existing triggers and return its allocated space.

```
TRUNCATE TABLE INVENTORY
  IGNORE DELETE TRIGGERS
  DROP STORAGE
  IMMEDIATE
```

- *Example 2:* Empty an unused inventory table regardless of any existing delete triggers but preserve its allocated space for later reuse.

```
TRUNCATE TABLE INVENTORY
   REUSE STORAGE
   IGNORE DELETE TRIGGERS
   IMMEDIATE
```

UPDATE

The UPDATE statement updates the values of specified columns in rows of a table, view or nickname, or the underlying tables, nicknames, or views of the specified fullselect.

Updating a row of a view updates a row of its base table, if no INSTEAD OF trigger is defined for the update operation on this view. If such a trigger is defined, the trigger will be executed instead. Updating a row using a nickname updates a row in the data source object to which the nickname refers.

The forms of this statement are:

- The *Searched* UPDATE form is used to update one or more rows (optionally determined by a search condition).
- The *Positioned* UPDATE form is used to update exactly one row (as determined by the current position of a cursor).

Invocation

An UPDATE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following authorities:

- UPDATE privilege on the target table, view, or nickname
- UPDATE privilege on each of the columns that are to be updated, including the columns of the BUSINESS_TIME period if a period-clause is specified
- CONTROL privilege on the target table, view, or nickname
- DATAACCESS authority

If a *row-fullselect* is included in the assignment, the privileges held by the authorization ID of the statement must include at least one of the following authorities for each referenced table, view, or nickname:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

For each table, view, or nickname referenced by a subquery, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of an UPDATE statement includes a reference to a column of the table, view, or nickname in the right side of the *assignment-clause*, or anywhere in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following authorities:

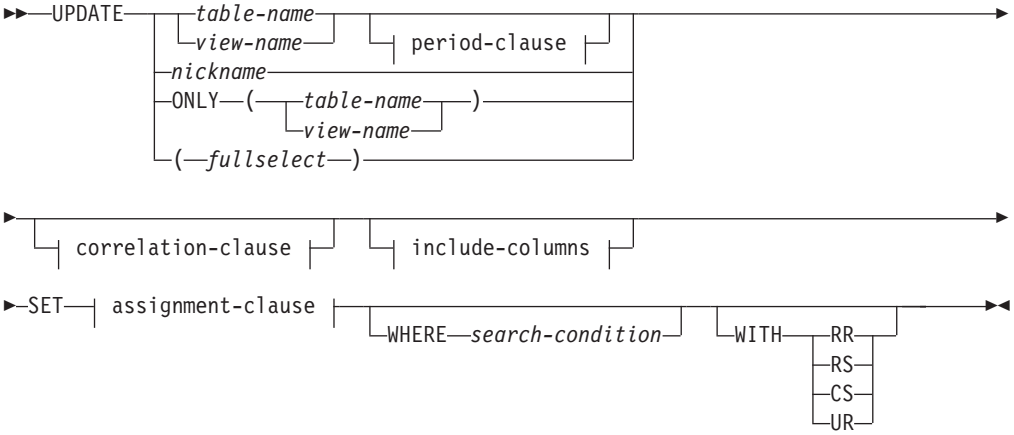
- SELECT privilege
- CONTROL privilege
- DATAACCESS authority

If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

GROUP privileges are not checked for static UPDATE statements.

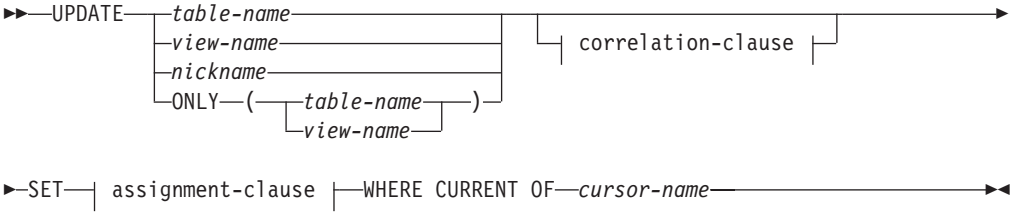
If the target of the update operation is a nickname, privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges that are required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax (searched-update)

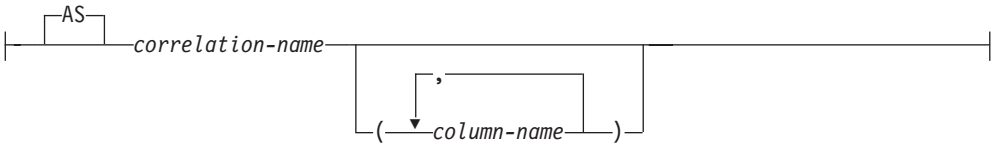


period-clause:

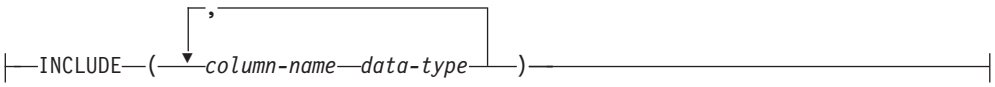
Syntax (positioned-update)



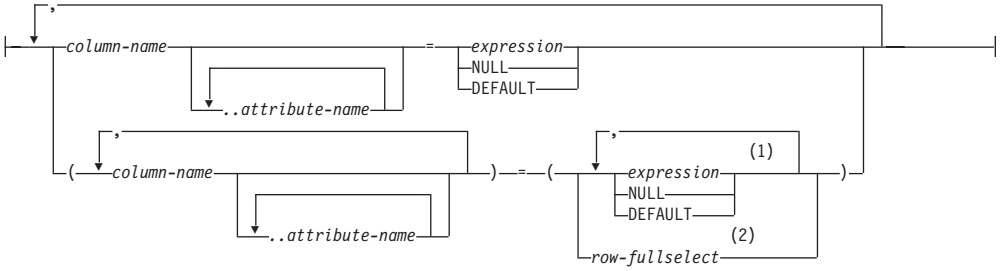
correlation-clause:



include-columns:



assignment-clause:



Notes:

1 The number of expressions, NULLs and DEFAULTs must match the number of column names.

2 The number of columns in the select list must match the number of column names.

Description

table-name, *view-name*, *nickname*, **or** *(fullselect)*

Identifies the object of the update operation. The name must identify one of the following objects:

- A table, view, or nickname described in the catalog at the current server
- A table or view at a remote server specified using a remote-object-name

The object must not be a catalog table, a view of a catalog table (unless it is one of the updatable SYSSTAT views), a system-maintained materialized query table, or a read-only view that has no INSTEAD OF trigger defined for its update operations.

If *table-name* is a typed table, rows of the table or any of its proper subtables may get updated by the statement. Only the columns of the specified table may be set or referenced in the WHERE clause. For a positioned UPDATE, the associated cursor must also have specified the same table, view or nickname in the FROM clause without using ONLY.

If the object of the update operation is a fullselect, the fullselect must be updatable, as defined in the "Updatable views" Notes item in the description of the CREATE VIEW statement.

If the object of the update operation is a nickname, the extended indicator variable values of DEFAULT and UNASSIGNED must not be used (SQLSTATE 22539).

For additional restrictions related to temporal tables and use of a view or *fullselect* as the target of the update operation, see "Considerations for a system-period temporal table" and "Considerations for an application-period temporal table" in the Notes section of this topic.

ONLY (*table-name***)**
Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

ONLY (*view-name***)**
Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

period-clause
Specifies that a period clause applies to the target of the update operation. If the target of the update operation is a view, the following conditions apply to the view:

- The FROM clause of the outer fullselect of the view definition must include a reference, directly or indirectly, to an application-period temporal table (SQLSTATE 42724M).
- An INSTEAD OF UPDATE trigger must not be defined for the view (SQLSTATE 428HY).

FOR PORTION OF BUSINESS_TIME
Specifies that the update only applies to row values for the portion of the period in the row that is specified by the period clause. The BUSINESS_TIME period must exist in the table (SQLSTATE 4274M).

FROM *value1* **TO** *value2*
Specifies that the update applies to rows for the period specified from *value1* up to *value2*. No rows are updated if *value1* is greater than or equal to *value2*, or if *value1* or *value2* is the null value (SQLSTATE 02000).

For the period specified with FROM *value1* TO *value2*, the BUSINESS_TIME period in a row in the target of the update is in any of the following states:

- **Overlaps the beginning** of the specified period if the value of the begin column is less than *value1* and the value of the end column is greater than *value1*.
- **Overlaps the end** of the specified period if the value of the end column is greater than or equal to *value2* and the value of the begin column is less than *value2*.
- Is **fully contained** within the specified period if the value for the begin column for BUSINESS_TIME is greater than or equal to *value1* and the value for the corresponding end column is less than or equal to *value2*.

- Is **partially contained** in the specified period if the row overlaps the beginning of the specified period or the end of the specified period, but not both.
- **Fully overlaps** the specified period if the period in the row overlaps the beginning and end of the specified period.
- Is **not contained** in the period if both columns of BUSINESS_TIME are less than or equal to *value1* or greater than or equal to *value2*.

If the BUSINESS_TIME period in a row is not contained in the specified period, the row is not updated. Otherwise, the update is applied based on how the values in the columns of the BUSINESS_TIME period overlap the specified period as follows:

- If the BUSINESS_TIME period in a row is fully contained within the specified period, the row is updated and the values of the begin column and end column of BUSINESS_TIME are unchanged.
- If the BUSINESS_TIME period in a row is partially contained in the specified period and overlaps the beginning of the specified period:
  - The row is updated. In the updated row, the value of the begin column is set to *value1* and the value of the end column is the original value of the end column.
  - A row is inserted using the original values from the row, except that the end column is set to *value1*.
- If the BUSINESS_TIME period in a row is partially contained in the specified period and overlaps the end of the specified period:
  - The row is updated. In the updated row, the value of the begin column is the original value of the begin column and the end column is set to *value2*.
  - A row is inserted using the original values from the row, except that the begin column is set to *value2*.
- If the BUSINESS_TIME period in a row fully overlaps the specified period:
  - The row is updated. In the updated row the value of the begin column is set to *value1* and the value of the end column is set to *value2*.
  - A row is inserted using the original values from the row, except that the end column is set to *value1*.
  - An additional row is inserted using the original values from the row, except that the begin column is set to *value2*.

value1 **and** *value2*

Each expression must return a value that has a date data type, timestamp data type, or a valid data type for a string representation of a date or timestamp (SQLSTATE 428HY). The result of each expression must be comparable to the data type of the columns of the specified period (SQLSTATE 42884). See the comparison rules described in "Assignments and comparisons".

Each expression can contain any of the following supported operands (SQLSTATE 428HY):

- Constant
- Special register
- Variable

- Scalar function whose arguments are supported operands (though user-defined functions and non-deterministic functions cannot be used)
- CAST specification where the cast operand is a supported operand
- Expression using arithmetic operators and operands

correlation-clause
Can be used within *search-condition* or *assignment-clause* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see "table-reference" in the description of "Subselect".

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the UPDATE statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the UPDATE statement.

column-name
Specifies a column of the intermediate result table of the UPDATE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

SET
Introduces the assignment of values to column names.

assignment-clause

column-name
Identifies a column to be updated. If extended indicator variables are not enabled, the *column-name* must identify an updatable column of the specified table, view, or nickname, or identify an INCLUDE column. The object ID column of a typed table is not updatable (SQLSTATE 428DZ). A column must not be specified more than once, unless it is followed by *..attribute-name* (SQLSTATE 42701).

If it specifies an INCLUDE column, the column name cannot be qualified.

For a Positioned UPDATE:

- If the *update-clause* was specified in the *select-statement* of the cursor, each column name in the *assignment-clause* must also appear in the *update-clause*.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL MIA or SQL92E was specified when the application was precompiled, the name of any updatable column may be specified.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL SAA1 was specified either explicitly or by default when the application was precompiled, no columns may be updated.

..attribute-name
Specifies the attribute of a structured type that is set (referred to as an

attribute assignment. The *column-name* specified must be defined with a user-defined structured type (SQLSTATE 428DP). The attribute-name must be an attribute of the structured type of *column-name* (SQLSTATE 42703). An assignment that does not involve the *..attribute-name* clause is referred to as a *conventional assignment*.

expression
Indicates the new value of the column. The expression is any expression of the type described in "Expressions". The expression cannot include an aggregate function except when it occurs within a scalar fullselect (SQLSTATE 42903).

An *expression* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated.

An expression cannot contain references to an INCLUDE column. If *expression* is a single host variable, the host variable can include an indicator variable that is enabled for extended indicator variables. If extended indicator variables are enabled, the extended indicator variable values of default (-5) or unassigned (-7) must not be used (SQLSTATE 22539) if either of the following statements is true:

- The expression is more complex than a single host variable with explicit casts
- The target column has data type of structured type

NULL
Specifies the null value and can only be specified for nullable columns (SQLSTATE 23502). NULL cannot be the value in an attribute assignment (SQLSTATE 429B9) unless it is specifically cast to the data type of the attribute.

DEFAULT
Specifies that the default value should be used based on how the corresponding column is defined in the table. The value that is inserted depends on how the column was defined.

- If the column was defined as a generated column based on an expression, the column value will be generated by the system, based on the expression.
- If the column was defined using the IDENTITY clause, the value is generated by the database manager.
- If the column was defined using the WITH DEFAULT clause, the value is set to the default defined for the column (see *default-clause* in "ALTER TABLE").
- If the column was defined using the NOT NULL clause and the GENERATED clause was not used, or the WITH DEFAULT clause was not used, or DEFAULT NULL was used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).
- If the column was defined using the ROW CHANGE TIMESTAMP clause, the value is generated by the database manager.

The only value that a generated column defined with the GENERATED ALWAYS clause can be set to is DEFAULT (SQLSTATE 428C9).

The DEFAULT keyword cannot be used as the value in an attribute assignment (SQLSTATE 429B9).

The DEFAULT keyword cannot be used as the value in an assignment for update on a nickname where the data source does not support DEFAULT syntax.

row-fullselect
A fullselect that returns a single row with the number of columns corresponding to the number of *column-name*s specified for assignment. The values are assigned to each corresponding *column-name*. If the result of the *row-fullselect* is no rows, then null values are assigned.

A *row-fullselect* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated. An error is returned if there is more than one row in the result (SQLSTATE 21000).

WHERE
Introduces a condition that indicates what rows are updated. You can omit the clause, give a search condition, or name a cursor. If the clause is omitted, all rows of the table, view or nickname are updated.

search-condition
Each *column-name* in the search condition, other than in a subquery, must name a column of the table, view or nickname. When the search condition includes a subquery in which the same table is the base object of both the UPDATE and the subquery, the subquery is completely evaluated before any rows are updated.

The search-condition is applied to each row of the table, view or nickname and the updated rows are those for which the result of the search-condition is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed only once, whereas a subquery with a correlated reference may have to be executed once for each row.

CURRENT OF *cursor-name*
Identifies the cursor to be used in the update operation. The *cursor-name* must identify a declared cursor, explained in "DECLARE CURSOR". The DECLARE CURSOR statement must precede the UPDATE statement in the program.

The specified table, view, or nickname must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR".)

When the UPDATE statement is executed, the cursor must be positioned on a row; that row is updated.

This form of UPDATE cannot be used (SQLSTATE 42828) if the cursor references:

- A view on which an INSTEAD OF UPDATE trigger is defined
- A view that includes an OLAP function in the select list of the fullselect that defines the view
- A view that is defined, either directly or indirectly, using the WITH ROW MOVEMENT clause

WITH
Specifies the isolation level at which the UPDATE statement is executed.

RR Repeatable Read

RS Read Stability

CS Cursor Stability

UR Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

Rules

- ***Triggers:*** UPDATE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the update values. If an update operation on a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.
- ***Assignment***: Update values are assigned to columns according to specific assignment rules.
- ***Validity***: The updated row must conform to any constraints imposed on the table (or on the base table of the view) by any unique index on an updated column.

  If a view is used that is not defined using WITH CHECK OPTION, rows can be changed so that they no longer conform to the definition of the view. Such rows are updated in the base table of the view and no longer appear in the view.

  If a view is used that is defined using WITH CHECK OPTION, an updated row must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW".
- ***Check constraint***: Update value must satisfy the check-conditions of the check constraints defined on the table.

  An UPDATE to a table with check constraints defined has the constraint conditions for each column updated evaluated once for each row that is updated. When processing an UPDATE statement, only the check constraints referring to the updated columns are checked.
- ***Referential integrity***: The value of the parent unique keys cannot be changed if the update rule is RESTRICT and there are one or more dependent rows. However, if the update rule is NO ACTION, parent unique keys can be updated as long as every child has a parent key by the time the update statement completes. A non-null update value of a foreign key must be equal to a value of the primary key of the parent table of the relationship.
- ***XML values:*** When an XML column value is updated, the new value must be a well-formed XML document (SQLSTATE 2200M).
- ***Security policy:*** If the identified table or the base table of the identified view is protected with a security policy, the session authorization ID must have the label-based access control (LBAC) credentials that allow:
  - Write access to all protected columns that are being updated (SQLSTATE 42512)
  - Write access for any explicit value provided for a DB2SECURITYLABEL column for security policies that were created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option (SQLSTATE 23523)

  - Read and write access to all rows that are being updated (SQLSTATE 42519)

  The session authorization ID must also have been granted a security label for write access for the security policy if an implicit value is used for a DB2SECURITYLABEL column (SQLSTATE 23523), which can happen when:

  - The DB2SECURITYLABEL column is not included in the list of columns that are to be updated (and so it will be implicitly updated to the security label for write access of the session authorization ID)
  - A value for the DB2SECURITYLABEL column is explicitly provided but the session authorization ID does not have write access for that value, and the security policy is created with the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option

- **Extended indicator variable usage**: If enabled, indicator variable values other than 0 (zero) through -7 must not be input (SQLSTATE 22010). Also, if enabled, the default and unassigned extended indicator variable values must not appear in contexts in which they are not supported (SQLSTATE 22539).
- **Extended indicator variables**: In the *assignment-clause* of an UPDATE statement, an *expression* that is a reference to a single host variable, or a host variable being explicitly cast can result in assigning an extended indicator variable value. Assigning an extended indicator variable-based value of unassigned has the effect of leaving the target column set to its current value, as if it had not been specified in the statement. Assigning an extended indicator variable-based value of default assigns the default value of the column. For information about default values of data types, see the description of the DEFAULT clause in "CREATE TABLE" on page 672.

  If a target column is not updatable (for example, a column in a view that is defined as an expression), then it must be assigned the extended indicator variable-based value of unassigned (SQLSTATE 42808).

  If the target column is a column defined as GENERATED ALWAYS, then it must be assigned the DEFAULT keyword, or the extended indicator variable-based values of default or unassigned (SQLSTATE 428C9).

  The UPDATE statement must not assign all target columns to an extended indicator variable-based value of unassigned (SQLSTATE 22540).

Notes

- If an update value violates any constraints, or if any other error occurs during the execution of the UPDATE statement, no rows are updated. The order in which multiple rows are updated is undefined.
- An update to a view defined using the WITH ROW MOVEMENT clause could cause a delete operation and an insert operation against the underlying tables of the view. For details, see the description of the CREATE VIEW statement.
- When an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows that qualified for the update operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. The SQLERRD(5) field contains the number of rows inserted, deleted, or updated by all activated triggers.
- Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful UPDATE statement. Until the locks are released, the updated row can only be accessed by the application process that performed the update (except for applications using the Uncommitted Read isolation level). For further information on locking, see the descriptions of the COMMIT, ROLLBACK, and LOCK TABLE statements.

- When updating the column distribution statistics for a typed table, the subtable that first introduced the column must be specified.
- Multiple attribute assignments on the same structured type column occur in the order specified in the SET clause and, within a parenthesized set clause, in left-to-right order.
- An attribute assignment invokes the mutator method for the attribute of the user-defined structured type. For example, the assignment `st..a1=x` has the same effect as using the mutator method in the assignment `st = st..a1(x)`.
- While a given column may be a target column in only one conventional assignment, a column may be a target column in multiple attribute assignments (but only if it is not also a target column in a conventional assignment).
- When an identity column defined as a distinct type is updated, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type before the computation.)
- To have DB2 generate a value on a SET statement for an identity column, use the DEFAULT keyword:

  ```
  SET NEW.EMPNO = DEFAULT
  ```

  In this example, NEW.EMPNO is defined as an identity column, and the value used to update this column is generated by DB2.
- For more information about consuming values of a generated sequence for an identity column, or about exceeding the maximum value for an identity column, see "INSERT".
- With partitioned tables, an UPDATE WHERE CURRENT OF *cursor-name* operation can move a row from one data partition to another. After this occurs, the cursor is no longer positioned on the row, and no further UPDATE WHERE CURRENT OF *cursor-name* modifications to that row are possible. The next row in the cursor can be fetched, however.
- For a column defined using the ROW CHANGE TIMESTAMP clause, the value is always changed on update of the row. If the column is not specified in the SET list explicitly, the database manager still generates a value for that row. The value is unique for each table partition within the database partition and is set to the approximate timestamp corresponding to the row update.
- **Extended indicator variables and update triggers**: If a target column has been assigned with an extended indicator variable-based value of unassigned, that column is not considered to have been updated. That column is treated as if it had not been specified in the OF *column-name* list of any update trigger defined on the target table.
- **Extended indicator variables and deferred error checks**: When extended indicator variables are enabled, validation that would otherwise be done in statement preparation, to recognize an update of a non-updatable column, is deferred until statement execution, except for column level update privilege checking of static UPDATE statements. Whether an error should be reported can be determined only during execution based on the indicator value. The checking of column level update privilege for static UPDATE statements continues to be performed during bind processing even when extended indicator variables are enabled.
- **Considerations for a system-period temporal table:** The target of the UPDATE statement must not be a fullselect that references a view in the FROM clause followed by a period specification for SYSTEM_TIME if the view is defined with the WITH CHECK OPTION and the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):

- A subquery that references a system-period temporal table (directly or indirectly)
- An invocation of an SQL routine that has a package associated with it
- An invocation of an external routine with a data access indication other than NO SQL

If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, an underlying target of the UPDATE statement must not be a system-period temporal table (SQLSTATE 51046), and the target of the UPDATE statement must not be a view defined with the WITH CHECK OPTION if the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):

- A subquery that references a system-period temporal table (directly or indirectly)
- An invocation of an SQL routine that has a package associated with it
- An invocation of an external routine with a data access indication other than NO SQL

When a row of a system-period temporal table is updated, the database manager updates the values of the row-begin and transaction-start-ID columns as follows:

- A row-begin column is assigned a value that is generated using a reading of the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to the row begin or transaction start-ID column in a table, or a row in a system-period temporal table is deleted. The database manager ensures uniqueness of the generated values for a row-begin column across transactions. The timestamp value might be adjusted to ensure that rows inserted into an associated history table have the end timestamp value greater than the begin timestamp value which can happen when a conflicting transaction is updating the same row in the system-period temporal table. The database configuration parameter **systime_period_adj** must be set to Yes for this adjustment in the timestamp value to occur. If multiple rows are updated within a single SQL transaction and an adjustment is not needed, the values for the row-begin column are the same for all the rows and are unique from the values generated for the column for another transaction.
- A transaction start-ID column is assigned a unique timestamp value per transaction or the null value The null value is assigned to the transaction start-ID column if the column is nullable and there is a row-begin column in the table for which the value did not need to be adjusted. Otherwise, the value is generated using a reading of the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to the row begin or transaction start-ID column in a table, or a row in a system-period temporal table is deleted. If multiple rows are updated within a single SQL transaction, the values for the transaction start-ID column are the same for all the rows and are unique from the values generated for the column for another transaction.

If the UPDATE statement has a search condition containing a correlated subquery that references historical rows (explicitly referencing the name of the history table name or implicitly through the use of a period specification in the FROM clause), the old version of the updated rows that are inserted as historical rows (into the history table if any) are potentially visible to update operations for the rows subsequently processed for the statement.

The target of an UPDATE statement cannot be a fullselect that references a view in the FROM clause followed by a period specification for SYSTEM_TIME if both of the following conditions are true (SQLSTATE 51046):

  - The view is defined with the WITH CHECK OPTION.
  - The view definition includes a WHERE clause containing one of the following syntax elements:
    - A subquery that references a system-period temporal table (directly or indirectly).
    - An invocation of an SQL routine that has a package associated with it.
    - An invocation of an external routine with a data access indication other than NO SQL.

  If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, the underlying target (direct or indirect) of the UPDATE statement cannot be a system-period temporal table (SQLSTATE 51046).

  If the CURRENT TEMPORAL SYSTEM_TIME special register is set to a non-null value, the target of an UPDATE statement cannot be a view defined with the WITH CHECK OPTION if the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):
  - A subquery that references a system-period temporal table (directly or indirectly).
  - An invocation of an SQL routine that has a package associated with it.
  - An invocation of an external routine with a data access indication other than NO SQL.

- **Considerations for a history table:** When a row of a system-period temporal table is updated, a historical copy of the row is inserted into the corresponding history table and the end timestamp of the historical row is captured in the form of a system determined value that corresponds to the time of the data change operation. The database manager assigns the value that is generated using a reading of the time-of-day clock during execution of the first data change statement in the transaction that requires a value to be assigned to the row begin or transaction start-ID column in a table, or a row in a system-period temporal table is deleted. The database manager ensures uniqueness of the generated values for an end column in a history table across transactions. The timestamp value might be adjusted to ensure that rows inserted into the history table have the end timestamp value greater than the begin timestamp value which can happen when a conflicting transaction is updating the same row in the system-period temporal table (SQLSTATE 01695). The database configuration parameter **systime_period_adj** must be set to Yes for this adjustment in the timestamp value to occur.

  For an update operation, the adjustment only affects the value for the end column corresponding to the row-end column in the history table associated with the system-period temporal table. Take these adjustments into consideration on subsequent references to the table whether there is a search for the transaction start time in the values for the columns corresponding to the row-begin and row-end columns of the period in the associated system-period temporal table.
- **Considerations for an application-period temporal table:** The target of the UPDATE statement must not be a fullselect that references a view in the FROM clause followed by a period specification for BUSINESS_TIME if the view is defined with the WITH CHECK OPTION and the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):
  - A subquery that references an application-period temporal table (directly or indirectly)
  - An invocation of an SQL routine that has a package associated with it

– An invocation of an external routine with a data access indication other than NO SQL

If the CURRENT TEMPORAL BUSINESS_TIME special register is set to a non-null value, the target of the UPDATE statement must not be a view defined with the WITH CHECK option if the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):

– A subquery that references an application-period temporal table (directly or indirectly)
– An invocation of an SQL routine that has a package associated with it
– An invocation of an external routine with a data access indication other than NO SQL

An UPDATE statement for an application-period temporal table that contains a FOR PORTION OF BUSINESS_TIME clause indicates between which two points in time that the specified updates are effective. When FOR PORTION OF BUSINESS_TIME is specified and the period value for a row, specified by the values of the row-begin column and row-end column, is only partially contained in the period specified from *value1* up to *value2*, the row is updated and one or two rows are automatically inserted to represent the portion of the row that is not changed. New values are generated for each generated column in an application-period temporal table for each row that is automatically inserted as a result of an update operation on the table. If a generated column is defined as part of a unique or primary key, parent key in a referential constraint, or unique index, it is possible that an automatic insert will violate a constraint or index in which case an error is returned.

When a row is inserted into an application-period temporal table that has either a primary key or unique constraint with the BUSINESS_TIME WITHOUT OVERLAPS clause defined, or a unique index with the BUSINESS_TIME WITHOUT OVERLAPS clause defined, if the period defined by the begin and end columns of the BUSINESS_TIME period overlap the period defined by the begin and end columns of the BUSINESS_TIME period for another row with the same unique constraint or unique index in the table, an error is returned.

The target of an UPDATE statement cannot be a fullselect that references a view in the FROM clause followed by a period specification for BUSINESS_TIME if both of the following conditions are true (SQLSTATE 51046):

– The view is defined with the WITH CHECK OPTION.
– The view definition includes a WHERE clause containing one of the following syntax elements:
  - A subquery that references an application-period temporal table (directly or indirectly).
  - An invocation of an SQL routine that has a package associated with it.
  - An invocation of an external routine with a data access indication other than NO SQL.

If the CURRENT TEMPORAL BUSINESS_TIME special register is set to a non-null value, the target of an UPDATE statement cannot be a view defined with the WITH CHECK OPTION if the view definition includes a WHERE clause containing one of the following syntax elements (SQLSTATE 51046):

– A subquery that references an application-period temporal table (directly or indirectly).
– An invocation of an SQL routine that has a package associated with it.
– An invocation of an external routine with a data access indication other than NO SQL.

When an application-period temporal table is the target of an UPDATE statement, the value in effect for the CURRENT TEMPORAL BUSINESS_TIME special register is not the null value, and the BUSTIMESENSITIVE bind option is set to YES, the following additional predicates are implicit:

```
     bt_begin <= CURRENT TEMPORAL BUSINESS_TIME
 AND bt_end > CURRENT TEMPORAL BUSINESS_TIME
```

where bt_begin and bt_end are the begin and end columns of the BUSINESS_TIME period of the target table of the UPDATE statement.

- **Considerations for application-period temporal tables and triggers:** When a row is updated and the FOR PORTION OF BUSINESS_TIME clause is specified, additional rows may be implicitly inserted to reflect any portion of the row that was not updated. Any existing update triggers are activated for the rows updated, and any existing insert triggers are activated for rows that are implicitly inserted.

Examples

- *Example 1:* Change the job (JOB) of employee number (EMPNO) '000290' in the EMPLOYEE table to 'LABORER'.

```
UPDATE EMPLOYEE
   SET JOB = 'LABORER'
   WHERE EMPNO = '000290'
```

- *Example 2:* Increase the project staffing (PRSTAFF) by 1.5 for all projects that department (DEPTNO) 'D21' is responsible for in the PROJECT table.

```
UPDATE PROJECT
   SET PRSTAFF = PRSTAFF + 1.5
   WHERE DEPTNO = 'D21'
```

- *Example 3:* All the employees except the manager of department (WORKDEPT) 'E21' have been temporarily reassigned. Indicate this by changing their job (JOB) to the null value and their pay (SALARY, BONUS, COMM) values to zero in the EMPLOYEE table.

```
UPDATE EMPLOYEE
  SET JOB=NULL, SALARY=0, BONUS=0, COMM=0
  WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
```

This statement could also be written as follows.

```
UPDATE EMPLOYEE
  SET (JOB, SALARY, BONUS, COMM) = (NULL, 0, 0, 0)
  WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
```

- *Example 4:* Update the salary and the commission column of the employee with employee number 000120 to the average of the salary and of the commission of the employees of the updated row's department, respectively.

```
UPDATE (SELECT EMPNO, SALARY, COMM,
  AVG(SALARY) OVER (PARTITION BY WORKDEPT),
  AVG(COMM) OVER (PARTITION BY WORKDEPT)
  FROM EMPLOYEE E) AS E(EMPNO, SALARY, COMM, AVGSAL, AVGCOMM)
SET (SALARY, COMM) = (AVGSAL, AVGCOMM)
WHERE EMPNO = '000120'
```

The previous statement is semantically equivalent to the following statement, but requires only one access to the EMPLOYEE table, whereas the following statement specifies the EMPLOYEE table twice.

```
UPDATE EMPLOYEE EU
  SET (EU.SALARY, EU.COMM)
  =
```

```
(SELECT AVG(ES.SALARY), AVG(ES.COMM)
  FROM EMPLOYEE ES
  WHERE ES.WORKDEPT = EU.WORKDEPT)
  WHERE EU.EMPNO = '000120'
```

- *Example 5:* In a C program display the rows from the EMPLOYEE table and then, if requested to do so, change the job (JOB) of certain employees to the new job keyed in.

```
EXEC SQL  DECLARE C1 CURSOR FOR
               SELECT *
                 FROM EMPLOYEE
                 FOR UPDATE OF JOB;

EXEC SQL  OPEN C1;

EXEC SQL  FETCH C1 INTO ...     ;
if ( strcmp (change, "YES") == 0 )
  EXEC SQL  UPDATE EMPLOYEE
              SET JOB = :newjob
              WHERE CURRENT OF C1;

EXEC SQL  CLOSE C1;
```

- *Example 6:* These examples mutate attributes of column objects.

  Assume that the following types and tables exist:

```
CREATE TYPE POINT AS (X INTEGER, Y INTEGER)
  NOT FINAL WITHOUT COMPARISONS
  MODE DB2SQL

CREATE TYPE CIRCLE AS (RADIUS INTEGER, CENTER POINT)
  NOT FINAL WITHOUT COMPARISONS
  MODE DB2SQL

CREATE TABLE CIRCLES (ID INTEGER, OWNER VARCHAR(50), C CIRCLE
```

  The following example updates the CIRCLES table by changing the OWNER column and the RADIUS attribute of the CIRCLE column where the ID is 999:

```
UPDATE CIRCLES
  SET OWNER = 'Bruce'
    C..RADIUS = 5
  WHERE ID = 999
```

  The following example transposes the X and Y coordinates of the center of the circle identified by 999:

```
UPDATE CIRCLES
  SET C..CENTER..X = C..CENTER..Y,
    C..CENTER..Y = C..CENTER..X
  WHERE ID = 999
```

  The following example is another way of writing both of the previous statements. This example combines the effects of both of the previous examples:

```
UPDATE CIRCLES
  SET (OWNER,C..RADIUS,C..CENTER..X,C..CENTER..Y) =
    ('Bruce',5,C..CENTER..Y,C..CENTER..X)
  WHERE ID = 999
```

- *Example 7:* Update the XMLDOC column of the DOCUMENTS table with DOCID '001' to the character string that is selected and parsed from the XMLTEXT table.

```
UPDATE DOCUMENTS SET XMLDOC =
  (SELECT XMLPARSE(DOCUMENT C1 STRIP WHITESPACE)
    FROM XMLTEXT WHERE TEXTID = '001')
WHERE DOCID = '001'
```

VALUES

The VALUES statement is a form of query.

The VALUES statement can be embedded in an application program or issued interactively.

VALUES INTO

The VALUES INTO statement produces a result table consisting of at most one row, and assigns the values in that row to host variables.

Invocation

This statement can be embedded only in an application program. It is an executable statement that cannot be dynamically prepared.

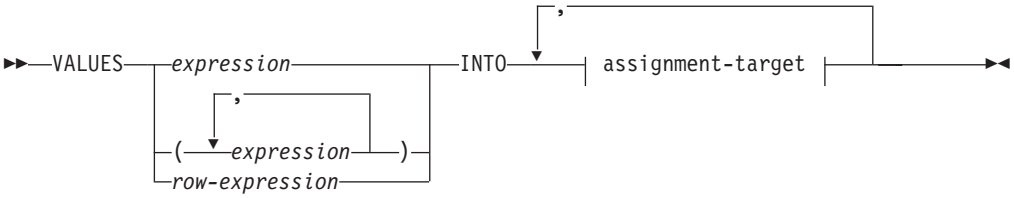
Authorization

The privileges held by the authorization ID of the statement must include any privileges that are necessary to execute each *expression* and *row-expression*.

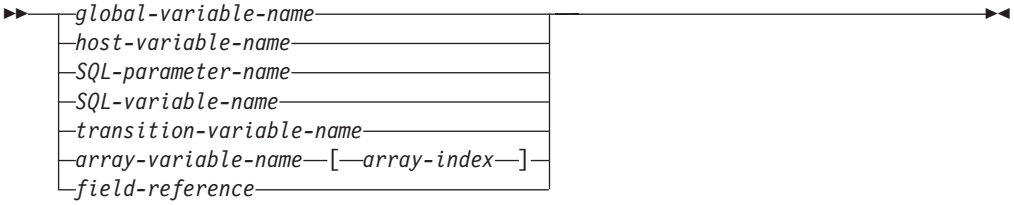
For each global variable used as an *assignment-target*, the privileges held by the authorization ID of the statement must include one of the following authorities:

- WRITE privilege on the global variable that is not defined in a module
- EXECUTE privilege on the module of the global variable that is defined in a module

Syntax



assignment-target



Description

VALUES
Introduces a single row consisting of one or more columns.

expression
An expression that defines a single value of a one column result table.

(expression,...)
One or more expressions that define the values for one or more columns of the result table.

row-expression
Specifies the new row of values. The *row-expression* is any row expression of the type described in "Row expressions". The *row-expression* must not include a column name.

INTO *assignment-target*
Identifies one or more targets for the assignment of output values.

The first value in the result row is assigned to the first target in the list, the second value to the second target, and so on. Each assignment to an *assignment-target* is made in sequence through the list. If an error occurs on any assignment, no value is assigned to any *assignment-target*.

When the data type of every *assignment-target* is not a row type, then the value 'W' is assigned to the SQLWARN3 field of the SQLCA if the number of *assignment-targets* is less than the number of result column values.

If the data type of an *assignment-target* is a row type, then there must be exactly one *assignment-target* specified (SQLSTATE 428HR), the number of columns must match the number of fields in the row type, and the data types of the columns of the fetched row must be assignable to the corresponding fields of the row type (SQLSTATE 42821).

If the data type of an *assignment-target* is an array element, then there must be exactly one *assignment-target* specified.

global-variable-name
Identifies the global variable that is the assignment target.

host-variable-name
Identifies the host variable that is the assignment target. For LOB output values, the target can be a regular host variable (if it is large enough), a LOB locator variable, or a LOB file reference variable.

SQL-parameter-name
Identifies the name parameter that is the assignment target.

SQL-variable-name
Identifies the SQL variable that is the assignment target. SQL variables must be declared before they are used.

transition-variable-name
Identifies the column to be updated in the transition row. A *transition-variable-name* must identify a column in the subject table of a trigger, optionally qualified by a correlation name that identifies the new value.

array-variable-name
Identifies an SQL variable, SQL parameter, or global variable of an array type.

[array-index]
An expression that specifies which element in the array will be the target of the assignment. For an ordinary array, the *array-index* expression must be assignable to INTEGER (SQLSTATE 428H1) and cannot be the null value. Its value must be between 1 and the maximum cardinality defined for the array (SQLSTATE 2202E). For an associative array, the *array-index* expression must be assignable to the index data type of the associative array (SQLSTATE 428H1) and cannot be the null value.

field-reference
Identifies the field within a row type value that is the assignment target. The *field-reference* must be specified as a qualified *field-name* where the qualifier identifies the row value in which the field is defined.

Rules

- Global variables cannot be assigned inside triggers that are not defined using a compound SQL (compiled) statement, functions that are not defined using a compound SQL (compiled) statement, methods, or compound SQL (inlined) statements (SQLSTATE 428GX).

Examples

- *Example 1:* This C example retrieves the value of the CURRENT PATH special register into a host variable.

```
EXEC SQL VALUES(CURRENT PATH)
         INTO :hvl;
```

- *Example 2:* This C example retrieves a portion of a LOB field into a host variable, exploiting the LOB locator for deferred retrieval.

```
EXEC SQL VALUES (substr(:locator1,35))
         INTO :details;
```

- *Example 3:* This C example retrieves the value of the SESSION_USER special register into a global variable.

```
EXEC SQL VALUES(SESSION_USER)
  INTO GV_SESS_USER;
```

WHENEVER

The WHENEVER statement specifies the action to be taken when a specified exception condition occurs.

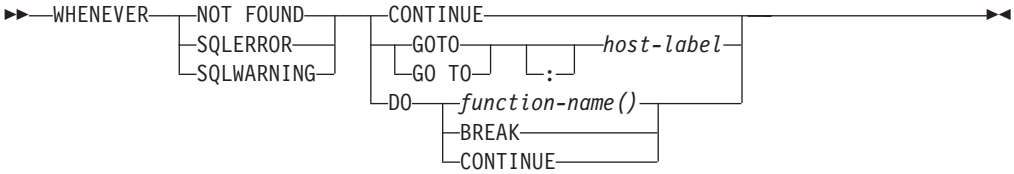
Invocation

This statement can only be embedded in an application program. It is not an executable statement. The statement is not supported in REXX.

Authorization

None required.

Syntax



Description

The NOT FOUND, SQLERROR, or SQLWARNING clause is used to identify the type of exception condition.

NOT FOUND
Identifies any condition that results in an SQLCODE of +100 or an SQLSTATE of '02000'.

SQLERROR
Identifies any condition that results in a negative SQLCODE.

SQLWARNING
Identifies any condition that results in a warning condition (SQLWARN0 is 'W'), or that results in a positive SQL return code other than +100.

The CONTINUE or GO TO clause is used to specify what is to happen when the identified type of exception condition exists.

CONTINUE
Causes the next sequential instruction of the source program to be executed.

GOTO or GO TO *host-label*
Causes control to pass to the statement identified by host-label. For host-label, substitute a single token, optionally preceded by a colon. The form of the token depends on the host language.

DO Causes additional action in the form of a function call, break statement, or continue statement to take place.

function-name()
Specifies the C function that is to be called. The function must have a void return value and cannot accept any arguments. The function name must end with set of parentheses "(" and ")". The name of the function is limited to 255 bytes.

The function name resolution takes place during the compilation of the C and C++ embedded SQL application. The DB2 precompiler does not resolve the function name.

BREAK
Specifies the C break statement. The C break statement exits thedo, for, switch, or while statement block.

CONTINUE
Specifies the C continue statement. The C continue statement passes control to the next iteration of the do, for, switch, or while statement block.

Notes

There are three types of WHENEVER statements:

- WHENEVER NOT FOUND
- WHENEVER SQLERROR
- WHENEVER SQLWARNING

Every executable SQL statement in a program is within the scope of one implicit or explicit WHENEVER statement of each type. The scope of a WHENEVER statement is related to the listing sequence of the statements in the program, not their execution sequence.

An SQL statement is within the scope of the last WHENEVER statement of each type that is specified before that SQL statement in the source program. If a WHENEVER statement of some type is not specified before an SQL statement, that SQL statement is within the scope of an implicit WHENEVER statement of that type in which CONTINUE is specified.

If the WHENEVER statement is not used, the default action is to continue processing if an error, warning, or exception condition occurs during execution.

The WHENEVER statement must be used before the SQL statements that you want to affect. Otherwise, the precompiler does not know that additional error-handling code is required for the executable SQL statements. You can have any combination of the three basic forms active at any time. The order in which you declare the three forms is not significant.

To avoid an infinite looping situation, ensure that you undo the WHENEVER handling before any SQL statements are executed inside the handler. You can undo the WHENEVER handling by using the WHENEVER SQLERROR CONTINUE statement.

The WHENEVER statement support for use of the DO *function-name()*, DO BREAK, or DO CONTINUE syntax is available in Version 9.7 Fix Pack 6 and later.

Example

In the following C example, if an error is produced, go to HANDLERR. If a warning code is produced, continue with the normal flow of the program. If no data is returned, go to ENDDATA.

```
EXEC SQL WHENEVER SQLERROR GOTO HANDLERR;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND GO TO ENDDATA;
```

The C example for use of the DO *function-name()*, DO BREAK, or DO CONTINUE syntax are:

```
/* DO function_name */
EXEC SQL WHENEVER SQLERROR DO perform_error_action();
EXEC SQL WHENEVER SQLWARNING DO perform_warning_action();
EXEC SQL WHENEVER NOT FOUND DO perform_notfound_action();

/* DO BREAK */
EXEC SQL WHENEVER SQLERROR DO BREAK;
EXEC SQL WHENEVER SQLWARNING DO BREAK;
EXEC SQL WHENEVER NOT FOUND DO BREAK;

/* DO CONTINUE */
EXEC SQL WHENEVER SQLERROR DO CONTINUE;
EXEC SQL WHENEVER SQLWARNING DO CONTINUE;
EXEC SQL WHENEVER NOT FOUND DO CONTINUE;
```

WHILE

The WHILE statement repeats the execution of a statement or group of statements while a specified condition is true.

Invocation

This statement can be embedded in an:

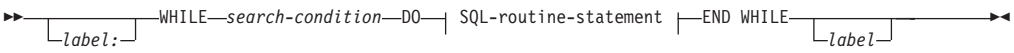
- SQL procedure definition
- Compound SQL (compiled) statement
- Compound SQL (inlined) statement

The compound statements can be embedded in an SQL procedure definition, SQL function definition, or SQL trigger definition. It is not an executable statement and cannot be dynamically prepared.

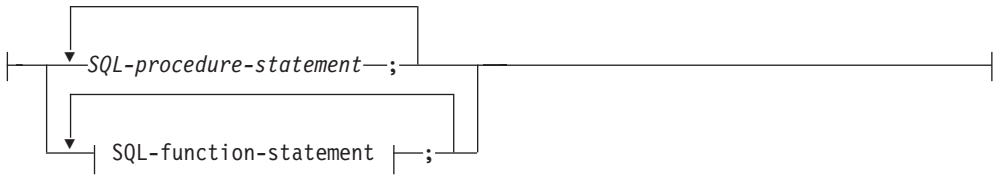
Authorization

No privileges are required to invoke the WHILE statement. However, the authorization ID of the statement must hold the necessary privileges to invoke the SQL statements and search condition that are embedded in the WHILE statement.

Syntax



SQL-routine-statement:



Description

label
Specifies the label for the WHILE statement. If the beginning label is specified, it can be specified in LEAVE and ITERATE statements. If the ending label is specified, it must be the same as the beginning label.

search-condition
Specifies a condition that is evaluated before each execution of the loop. If the condition is true, the SQL-procedure-statements in the loop are processed.

SQL-procedure-statement
Specifies the SQL statements to execute within the loop. *SQL-procedure-statement* is only applicable when in the context of an SQL procedure or compound SQL (compiled) statement. See *SQL-procedure-statement* in "Compound SQL (compiled)" statement.

SQL-function-statement
Specifies the SQL statements to execute within the loop. *SQL-function-statement*

is only applicable in an SQL function or a compound SQL (inlined) statement which can be embedded in an SQL trigger, SQL function or SQL method. See *SQL-function-statement* in "FOR".

Example

This example uses a WHILE statement to iterate through FETCH and SET statements. While the value of SQL variable *v_counter* is less than half of number of employees in the department identified by the IN parameter *deptNumber*, the WHILE statement continues to perform the FETCH and SET statements. When the condition is no longer true, the flow of control leaves the WHILE statement and closes the cursor.

```
CREATE PROCEDURE DEPT_MEDIAN
  (IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
  LANGUAGE SQL
  BEGIN
    DECLARE v_numRecords INTEGER DEFAULT 1;
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE c1 CURSOR FOR
      SELECT CAST(salary AS DOUBLE)
        FROM staff
        WHERE DEPT = deptNumber
        ORDER BY salary;
    DECLARE EXIT HANDLER FOR NOT FOUND
      SET medianSalary = 6666;
    SET medianSalary = 0;
    SELECT COUNT(*) INTO v_numRecords
      FROM staff
      WHERE DEPT = deptNumber;
    OPEN c1;
    WHILE v_counter < (v_numRecords / 2 + 1) DO
      FETCH c1 INTO medianSalary;
      SET v_counter = v_counter + 1;
    END WHILE;
    CLOSE c1;
  END
```

Appendix A. Overview of the DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command-line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss. English and translated DB2 Version 10.1 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg27009474.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 37. DB2 technical information

Name	Form Number	Available in print	Last updated
Administrative API Reference	SC27-3864-00	Yes	April, 2012
Administrative Routines and Views	SC27-3865-01	No	January, 2013
Call Level Interface Guide and Reference Volume 1	SC27-3866-01	Yes	January, 2013
Call Level Interface Guide and Reference Volume 2	SC27-3867-01	Yes	January, 2013
Command Reference	SC27-3868-01	Yes	January, 2013
Database Administration Concepts and Configuration Reference	SC27-3871-01	Yes	January, 2013
Data Movement Utilities Guide and Reference	SC27-3869-01	Yes	January, 2013
Database Monitoring Guide and Reference	SC27-3887-01	Yes	January, 2013
Data Recovery and High Availability Guide and Reference	SC27-3870-01	Yes	January, 2013
Database Security Guide	SC27-3872-01	Yes	January, 2013
DB2 Workload Management Guide and Reference	SC27-3891-01	Yes	January, 2013
Developing ADO.NET and OLE DB Applications	SC27-3873-01	Yes	January, 2013
Developing Embedded SQL Applications	SC27-3874-01	Yes	January, 2013
Developing Java Applications	SC27-3875-01	Yes	January, 2013

Table 37. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-3876-00	No	April, 2012
Developing RDF Applications for IBM Data Servers	SC27-4462-00	Yes	January, 2013
Developing User-defined Routines (SQL and External)	SC27-3877-01	Yes	January, 2013
Getting Started with Database Application Development	GI13-2046-01	Yes	January, 2013
Getting Started with DB2 Installation and Administration on Linux and Windows	GI13-2047-00	Yes	April, 2012
Globalization Guide	SC27-3878-00	Yes	April, 2012
Installing DB2 Servers	GC27-3884-01	Yes	January, 2013
Installing IBM Data Server Clients	GC27-3883-00	No	April, 2012
Message Reference Volume 1	SC27-3879-01	No	January, 2013
Message Reference Volume 2	SC27-3880-01	No	January, 2013
Net Search Extender Administration and User's Guide	SC27-3895-01	No	January, 2013
Partitioning and Clustering Guide	SC27-3882-01	Yes	January, 2013
Preparation Guide for DB2 10.1 Fundamentals Exam 610	SC27-4540-00	No	January, 2013
Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611	SC27-4541-00	No	January, 2013
pureXML Guide	SC27-3892-01	Yes	January, 2013
Spatial Extender User's Guide and Reference	SC27-3894-00	No	April, 2012
SQL Procedural Languages: Application Enablement and Support	SC27-3896-01	Yes	January, 2013
SQL Reference Volume 1	SC27-3885-01	Yes	January, 2013
SQL Reference Volume 2	SC27-3886-01	Yes	January, 2013
Text Search Guide	SC27-3888-01	Yes	January, 2013
Troubleshooting and Tuning Database Performance	SC27-3889-01	Yes	January, 2013

Table 37. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
Upgrading to DB2 Version 10.1	SC27-3881-01	Yes	January, 2013
What's New for DB2 Version 10.1	SC27-3890-01	Yes	January, 2013
XQuery Reference	SC27-3893-01	No	January, 2013

Table 38. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
DB2 Connect Installing and Configuring DB2 Connect Personal Edition	SC27-3861-00	Yes	April, 2012
DB2 Connect Installing and Configuring DB2 Connect Servers	SC27-3862-01	Yes	January, 2013
DB2 Connect User's Guide	SC27-3863-01	Yes	January, 2013

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.
For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

Documentation for other versions of DB2 products is found in separate information centers on ibm.com®.

About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v10r1.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

Before you begin

A DB2 Version 10.1 Information Center must already be installed. For details, see the "Installing the DB2 Information Center using the DB2 Setup wizard" topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates update existing Information Center features and languages. One benefit of automatic updates is that the Information Center is unavailable for a shorter time compared to during a manual update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates can be used to update existing Information Center features and languages. Automatic updates reduce the downtime during the update process, however you must use the manual process when you want to add features or languages. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process. In the automatic update process the Information Center incurs an outage to restart the Information Center after the update only.

This topic details the process for automatic updates. For manual update instructions, see the "Manually updating the DB2 Information Center installed on your computer or intranet server" topic.

Procedure

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
   a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V10.1` directory.
   b. Navigate from the installation directory to the `doc/bin` directory.
   c. Run the `update-ic` script:

```
update-ic
```

2. On Windows operating systems,
   a. Open a command window.
   b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `<Program Files>\IBM\DB2 Information Center\Version 10.1` directory, where `<Program Files>` represents the location of the Program Files directory.
   c. Navigate from the installation directory to the `doc\bin` directory.
   d. Run the `update-ic.bat` file:

      ```
      update-ic.bat
      ```

Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in `doc\eclipse\configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

About this task

Updating your locally installed *DB2 Information Center* manually requires that you:

1. Stop the *DB2 Information Center* on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

   Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system by using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.
   If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.
3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

Note: On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

Procedure

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
   - On Windows, click **Start** > **Control Panel** > **Administrative Tools** > **Services**. Then right-click **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv10 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the *Program_Files*\IBM\DB2 Information Center\Version 10.1 directory, where *Program_Files* represents the location of the Program Files directory.
     c. Navigate from the installation directory to the doc\bin directory.
     d. Run the help_start.bat file:
        ```
        help_start.bat
        ```
   - On Linux:
     a. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the /opt/ibm/db2ic/V10.1 directory.
     b. Navigate from the installation directory to the doc/bin directory.
     c. Run the help_start script:
        ```
        help_start
        ```

   The systems default Web browser opens to display the stand-alone Information Center.
3. Click the **Update** button ( ). (JavaScript must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check that the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
   - On Windows, navigate to the doc\bin directory within the installation directory, and run the help_end.bat file:
     ```
     help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely stop the processes that were started with the help_start batch file. Do not use Ctrl-C or any other method to stop help_start.bat.
   - On Linux, navigate to the doc/bin directory within the installation directory, and run the help_end script:
     ```
     help_end
     ```

     Note: The help_end script contains the commands required to safely stop the processes that were started with the help_start script. Do not use any other method to stop the help_start script.
7. Restart the *DB2 Information Center*.

- On Windows, click **Start** > **Control Panel** > **Administrative Tools** > **Services**. Then right-click **DB2 Information Center** service and select **Start**.
- On Linux, enter the following command:

```
/etc/init.d/db2icdv10 start
```

Results

The updated *DB2 Information Center* displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 database products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click the title.

"pureXML" in *pureXML Guide*
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation
Troubleshooting information can be found in the *Troubleshooting and Tuning Database Performance* or the Database fundamentals section of the *DB2 Information Center*, which contains:

- Information about how to isolate and identify problems with DB2 diagnostic tools and utilities.
- Solutions to some of the most common problem.
- Advice to help solve other problems you might encounter with your DB2 database products.

IBM Support Portal
See the IBM Support Portal if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the IBM Support Portal at http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

Terms and conditions

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Trademarks: IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

aliases
  adding comments to catalog 289
  CREATE ALIAS statement 343
  dropping 969
ALLOCATE CURSOR statement 24
ALTER AUDIT POLICY statement 26
ALTER BUFFERPOOL statement 29
ALTER DATABASE PARTITION GROUP statement 32
ALTER DATABASE statement
  details 36
ALTER EVENT MONITOR statement
  details 41
ALTER FUNCTION statement 46
ALTER HISTOGRAM TEMPLATE statement 50
ALTER INDEX statement 52
ALTER MASK statement 53
ALTER METHOD statement 54
ALTER NICKNAME statement 64
ALTER NODEGROUP statement
  see ALTER DATABASE PARTITION GROUP statement 32
ALTER PACKAGE statement 73
ALTER PERMISSION statement 76
ALTER PROCEDURE (External) statement 77
ALTER PROCEDURE (Sourced) statement 80
ALTER PROCEDURE (SQL) statement 82
ALTER SCHEMA statement 84
ALTER SECURITY LABEL COMPONENT statement 86
ALTER SECURITY POLICY statement 89
ALTER SEQUENCE statement 93
ALTER SERVER statement 97
ALTER SERVICE CLASS statement 100
ALTER STOGROUP statement
  details 110
ALTER TABLE statement
  details 114
ALTER TABLESPACE statement
  details 179
ALTER THRESHOLD statement 194
ALTER TRIGGER statement 207
ALTER TRUSTED CONTEXT statement 208
ALTER TYPE (Structured) statement 216
ALTER USAGE LIST statement 223
ALTER USER MAPPING statement 225
ALTER VIEW statement
  details 227
ALTER WORK ACTION SET statement 229
ALTER WORK CLASS SET statement 243
ALTER WORKLOAD statement
  details 249
ALTER WRAPPER statement 265
ALTER XSROBJECT statement 267
ambiguous cursors 928
arithmetic
  parameter markers 1130
assembler application host variables 1011
ASSOCIATE LOCATORS statement 268
ASUTIME
  CREATE FUNCTION (external scalar) statement 436
  CREATE FUNCTION (external table) statement 464
  CREATE PROCEDURE (external) statement 604
ASUTIME *(continued)*
  CREATE PROCEDURE (SQL) statement 626
AUDIT statement 270
authorization IDs
  granting control
    database operations 1039
    indexes 1049
  granting schema privileges 1063
  public control on index 1049
  revoking authorities 1154

B

BEGIN DECLARE SECTION statement 274
BIGINT data type
  CREATE TABLE statement 672
binary large objects (BLOBs)
  tables 672
binding
  GRANT statement 1053
  revoking BIND privilege 1166
BLOB data type
  CREATE TABLE statement 672
buffer pools
  creating 350
  dropping 969
  page size 350
  setting size 29, 350

C

caching
  EXECUTE statement 1003
CALL statement
  details 276
CASCADE delete rule 672
CASE statement
  details 284
catalogs
  COMMENT statement 289
CHAR VARYING data type 672
CHARACTER data type 672
character strings
  SQL statement creation 1011
CHARACTER VARYING data type 672
check constraints
  ALTER TABLE statement 114
  CREATE TABLE statement 672
  INSERT statement 1091
CLOB data type
  columns 672
CLOSE statement
  details 287
closed state
  cursors 1122
coded character set identifier (CCSID)
  CREATE TABLE statement 672
  DECLARE GLOBAL TEMPORARY TABLE statement 934
COLLID
  CREATE FUNCTION (external scalar) statement 436

COLLID *(continued)*
CREATE FUNCTION (external table) statement 464
CREATE PROCEDURE (external) statement 604
CREATE PROCEDURE (SQL) statement 626
columns
adding
ALTER TABLE statement 114
comment additions in catalog 289
constraints
names 672
granting add privileges 1078
index keys 545
names
INSERT statement 1091
null values
ALTER TABLE statement 114
updating 1321
values
inserting 1091
COMMENT statement 289
comments
catalog table 289
SQL
static statements 10
SQL static statements 13
COMMIT statement
details 300
compilation
conditional (SQL) 14
compiled compound statement
details 312
compound SQL statements
embedded 308
inlined 303
overview 302
concurrency
LOCK TABLE statement 1106
condition handlers
declaring 312
conditional compilation
SQL 14
CONNECT statement
type 1 329
type 2 336
constraints
adding comments to catalog 289
adding with ALTER TABLE statement 114
dropping 114
containers
CREATE TABLESPACE statement 752
conventions
highlighting x
conversion
character string to executable SQL 1011
CREATE ALIAS statement 343
CREATE AUDIT POLICY statement 347
CREATE BUFFERPOOL statement 350
CREATE DATABASE PARTITION GROUP statement 354
CREATE DISTINCT TYPE statement
see CREATE TYPE statement, distinct type 820
CREATE EVENT MONITOR (activities) statement 376
CREATE EVENT MONITOR (change history) statement 387
CREATE EVENT MONITOR (locking) statement 394
CREATE EVENT MONITOR (package cache) statement 400
CREATE EVENT MONITOR (statistics) statement 407
CREATE EVENT MONITOR (threshold violations) statement 419
CREATE EVENT MONITOR (unit of work) statement 430
CREATE EVENT MONITOR statement 356
CREATE FUNCTION MAPPING statement 526
CREATE FUNCTION statement
external scalar 436
external table 464
OLE external table 484
overview 435
sourced 495
SQL row 509
SQL scalar 509
SQL table 509
template 495
CREATE GLOBAL TEMPORARY TABLE statement
details 530
CREATE HISTOGRAM TEMPLATE statement 543
CREATE INDEX EXTENSION statement 566
CREATE INDEX statement
details 545
CREATE MASK statement 572
CREATE METHOD statement
details 578
CREATE MODULE statement 584
CREATE NICKNAME statement
details 586
CREATE NODEGROUP statement 354
CREATE PERMISSION statement 599
CREATE PROCEDURE statement
CASE statement 284
compound SQL 312
compound SQL (inlined) statement 303
condition handlers 312
DECLARE statement 312
external 604
FOR statement 1030
GET DIAGNOSTICS statement 1034
GOTO statement 1037
handler statement 312
IF statement 1087
ITERATE statement 1102
LEAVE statement 1104
LOOP statement 1108
overview 603
REPEAT statement 1147
RETURN statement 1152
SIGNAL statement 1303
sourced 620
SQL 626
variables 312
WHILE statement 1344
CREATE ROLE statement
details 636
CREATE SCHEMA statement 637
CREATE SECURITY LABEL COMPONENT statement 640
CREATE SECURITY LABEL statement 643
CREATE SECURITY POLICY statement 645
CREATE SEQUENCE statement 647
CREATE SERVER statement 664
CREATE SERVICE CLASS statement 653
CREATE STOGROUP statement
details 668
CREATE SYNONYM statement 671
CREATE TABLE statement
details 672
CREATE TABLESPACE statement
details 752

CREATE THRESHOLD statement
details 767
CREATE TRANSFORM statement
details 784
CREATE TRIGGER statement 788
CREATE TRUSTED CONTEXT statement
details 803
CREATE TYPE MAPPING statement
details 856
CREATE TYPE statement
array type 811
details 810
distinct type 820
row type 828
structured type 833
CREATE USAGE LIST statement 863
CREATE USER MAPPING statement
details 867
CREATE VARIABLE statement 869
CREATE VIEW statement 879
CREATE WORK ACTION SET statement 894
CREATE WORK CLASS SET statement 903
CREATE WORKLOAD statement
details 908
CREATE WRAPPER statement
details 926
CURRENT DECFLOAT ROUNDING MODE special register
SET CURRENT DECFLOAT ROUNDING MODE statement 1209
CURRENT DEGREE special register
SET CURRENT DEGREE statement 1212
CURRENT EXPLAIN MODE special register
SET CURRENT EXPLAIN MODE statement 1214
CURRENT EXPLAIN SNAPSHOT special register
SET CURRENT EXPLAIN SNAPSHOT statement 1217
CURRENT FUNCTION PATH special register
SET CURRENT FUNCTION PATH statement 1278
SET CURRENT PATH statement 1278
SET PATH statement 1278
CURRENT IMPLICIT XMLPARSE OPTION special register
SET CURRENT IMPLICIT XMLPARSE OPTION statement 1221
CURRENT ISOLATION special register
SET CURRENT ISOLATION statement 1222
CURRENT OPTIMIZATION PROFILE special register
SET CURRENT OPTIMIZATION PROFILE statement 1233
SET CURRENT TEMPORAL BUSINESS_TIME statement 1249
SET CURRENT TEMPORAL SYSTEM_TIME statement 1251
CURRENT PATH special register
SET CURRENT FUNCTION PATH statement 1278
SET CURRENT PATH statement 1278
SET PATH statement 1278
CURRENT QUERY OPTIMIZATION special register
SET CURRENT QUERY OPTIMIZATION statement 1242
CURRENT REFRESH AGE special register
SET CURRENT REFRESH AGE statement 1245
cursors
active set association 1122
ambiguous 928
closed state 1122
current row 1019
DECLARE CURSOR statement 928
declaring
SQL statement syntax 928
cursors *(continued)*
deleting 947
location in table as result of FETCH statement 1019
moving position using FETCH 1019
names
allocating 24
opening 1122
preparing for application use 1122
read-only
conditions 928
result table relationship 928
units of work
conditional states 928
terminating for 1195
updatable
determining 928
WITH HOLD
lock clause of COMMIT statement 300

D

data
integrity
locks 1106
data types
abstract 216, 833
ALTER TYPE statement 216
CREATE TYPE (structured) statement 833
declared 312
distinct
CREATE TYPE (distinct) statement 820
structured
ALTER TYPE (structured) statement 216
CREATE TYPE (structured) statement 833
user-defined
CREATE TYPE (distinct) statement 820
database authorities
granting
GRANT (database authorities) statement 1039
database partition groups
adding comments to catalog 289
adding partitions 32
creating 354
distribution map creation 354
dropping partitions 32
database-managed space (DMS)
table spaces
CREATE TABLESPACE statement 752
databases
accessing
granting authority 1039
CREATE TABLESPACE statement 752
DB2 Information Center
updating 1351, 1352
versions 1350
db2nodes.cfg file
ALTER DATABASE PARTITION GROUP statement 32
CONNECT (type 1) statement 329
CREATE DATABASE PARTITION GROUP statement 354
DB2SECURITYLABEL data type
CREATE TABLE statement 672
DBADM (database administration) authority
granting 1039
DBCLOB data type
CREATE TABLE statement 672
declarations
inserting into program 1089

DECLARE CURSOR statement
details 928
DECLARE GLOBAL TEMPORARY TABLE statement
details 934
DECLARE statements
BEGIN DECLARE SECTION statement 274
compound SQL 312
END DECLARE SECTION statement 1002
deletable views
overview 879
DELETE statement
details 947
dependent objects
DROP statement 969
deprecated functionality
SQL statements
ALTER DATABASE 36
DESCRIBE INPUT statement 958
DESCRIBE OUTPUT statement 962
DESCRIBE statement
details 957
prepared statements
DESCRIBE INPUT statement 958
DESCRIBE OUTPUT statement 962
DISCONNECT statement 966
distinct types
CREATE TYPE (distinct) statement 820
DROP statement 969
documentation
overview 1347
PDF files 1348
printed 1348
terms and conditions of use 1355
DROP statement
details 969
transforms 969
dynamic SQL
compound statements 303
cursors
DECLARE CURSOR statement 10, 11
DESCRIBE INPUT statement 958
DESCRIBE OUTPUT statement 962
EXECUTE IMMEDIATE statement
details 1011
EXECUTE statement
details 1003
invoking SQL statements 10, 11
FETCH statement
details 1019
invoking SQL statements 10, 11
invoking statements 10, 11
OPEN statement 10, 11
PREPARE statement
details 1130
invoking SQL statements 10, 11
using DESCRIBE 958, 962

E

embedded SQL applications
character string format statements 1011
EXECUTE IMMEDIATE statement 1011
overview 10
END DECLARE SECTION statement 1002
error conditions x
error messages
column masks 53, 572
error messages *(continued)*
return codes 10, 12
row permissions 76, 599
triggers
execution 788
typed tables 207
errors
cursors 1122
FETCH statement 1019
UPDATE statement 1321
event monitors
CREATE EVENT MONITOR statement 356
DROP statement 969
FLUSH EVENT MONITOR statement 1024
SET EVENT MONITOR STATE statement 1255
exception tables
SET INTEGRITY statement 1257
EXCLUSIVE MODE connection 329
executable SQL statements 10, 11, 12
EXECUTE IMMEDIATE statement
details 1011
embedded 10, 11
EXECUTE statement
details 1003
embedded 10, 11
EXPLAIN statement
details 1014

F

FETCH statement
cursor prerequisites for executing 1019
details 1019
FLOAT data type
CREATE TABLE statement 672
FLUSH BUFFERPOOLS statement 1023
FLUSH EVENT MONITOR statement 1024
FLUSH FEDERATED CACHE statement 1025
FLUSH OPTIMIZATION PROFILE CACHE statement 1027
FLUSH PACKAGE CACHE statement 1029
FOR statement 1030
foreign keys
adding 114
constraint names 672
dropping 114
FREE LOCATOR statement 1033
FROM clause
DELETE statement 947
fullselect
CREATE VIEW statement 879
function designator syntax element 20
functions
adding comments to catalog 289
templates
details 526
transformation 784

G

generated columns
CREATE TABLE statement 672
GET DIAGNOSTICS statement 1034
global variables
references 17
GOTO statement
details 1037

GRANT statement
database authorities 1039
exemptions 1044
global variable privileges 1047
index privileges 1049
nickname privileges 1078
package privileges 1053
roles 1056
routine privileges 1059
schema privileges 1063
security labels 1066
sequence privileges 1069
server privileges 1072
SETSESSIONUSER privilege 1074
table privileges 1078
table space privileges 1076
view privileges 1078
workload privileges 1084
XSR object privileges 1086
GRAPHIC data type
CREATE TABLE statement 672

H

hashing on partition keys 672
help
SQL statements 1350
host variables
assigning values from a row
SELECT INTO statement 1202
VALUES INTO statement 1338
BEGIN DECLARE SECTION statement 274
declaring
BEGIN DECLARE SECTION statement 274
cursors 928
END DECLARE SECTION statement 1002
embedded SQL statements 10, 12
END DECLARE SECTION statement 1002
EXECUTE IMMEDIATE statement 1011
FETCH statement 1019
inserting in rows 1091
linking active set with cursor 1122
parameter marker substitution 1003
REXX applications 274
statement strings 1130

I

identity columns
CREATE TABLE statement 672
IF statement
SQL 1087
implicit connections
CONNECT statement 329
implicit schemas
GRANT (database authorities) statement 1039
REVOKE (database authorities) statement 1154
INCLUDE statement
details 1089
index over XML data
CREATE INDEX statement
details 545
indexes
catalog specification comments 289
correspondence to inserted row values 1091
dropping 969
indexes *(continued)*
granting control 1049, 1078
names
primary key constraint 672
unique constraint 672
primary key 114
privileges
revoking 1162
renaming 1143
unique key 114
inoperative triggers 207, 788
inoperative views 879
INSERT statement 1091
insertable views
creating 879
INTEGER data type
CREATE TABLE statement 672
integrity constraints 289
isolation levels
DELETE statement 947
INSERT statement 1091
SELECT statement 1202
UPDATE statement 1321
ITERATE statement
details 1102

J

joins
CREATE TABLE statement 672

L

labels
GOTO statement 1037
SQL procedures 18
LBAC
ALTER SECURITY LABEL COMPONENT statement 86
ALTER SECURITY POLICY statement 89
CREATE SECURITY LABEL COMPONENT statement 640
CREATE SECURITY LABEL statement 643
CREATE SECURITY POLICY statement 645
GRANT (exemption) statement 1044
GRANT (security label) statement 1066
REVOKE (exemption) statement 1158
REVOKE (security label) statement 1177
rule exemptions
GRANT (exemption) statement 1044
REVOKE (exemption) statement 1158
security label components
ALTER SECURITY LABEL COMPONENT statement 86
CREATE SECURITY LABEL COMPONENT statement 640
security labels
ALTER SECURITY LABEL COMPONENT statement 86
CREATE SECURITY LABEL COMPONENT statement 640
CREATE SECURITY LABEL statement 643
GRANT (security label) statement 1066
REVOKE (security label) statement 1177
security policies
ALTER SECURITY POLICY statement 89
CREATE SECURITY POLICY statement 645

LEAVE statement
  details 1104
loads
  granting database authority 1039
locators
  ASSOCIATE LOCATORS statement 268
  FREE LOCATOR statement 1033
LOCK TABLE statement
  details 1106
locks
  COMMIT statement 300
  INSERT statement 1091
  LOCK TABLE statement 1106
  restricting access 1106
  terminating for unit of work 1195
  UPDATE statement 1321
logs
  creating tables without initial logging 672
LOOP statement
  SQL 1108

M

masks
  ALTER MASK statement 53
  CREATE MASK statement 572
MERGE statement 1110
method designator syntax element 20
MODE keyword 1106
modules
  altering 56
  creating 584
MQTs
  defining 672
  REFRESH TABLE statement 1136

N

nicknames
  details 586
  privileges
    granting 1078
    revoking 1187
NO ACTION delete rule 672
nonexecutable SQL statements
  invoking 10
  precompiler requirements 10
NOT FOUND clause
  WHENEVER statement 1341
notices 1357

O

object identifiers (OIDs)
  columns
    overview 672
  CREATE TABLE statement 672
  CREATE VIEW statement 879
OID
  see object identifiers (OIDs) 672
OPEN statement
  details 1122

P

packages
  ALTER TABLE statement 114
  authority to create 1039
  catalog comments 289
  COMMIT statement effect on cursors 300
  deleting 969
  privileges
    granting 1053
    revoking using REVOKE (package privileges) statement 1166
    revoking using REVOKE (table, view, or nickname privileges) statement 1187
parameter markers
  EXECUTE statement 1003
  OPEN statement 1122
  password rules 1130
  PREPARE statement 1130
  typed 1130
  untyped 1130
partitioning keys
  adding 114
  defining when creating tables 672
  dropping 114
partitioning maps
  creating for database partition groups 354
performance
  partitioning key recommendation 672
permissions
  ALTER PERMISSION statement 76
  CREATE PERMISSION statement 599
PIPE statement 1128
positional updating of columns by row 1321
precompilation
  external text files 1089
  INCLUDE statement 1089
  non-executable SQL statements 10
  SQLCA 1089
  SQLDA 1089
PREPARE statement
  details 1130
  dynamically declaring 1130
  embedded 10, 11
  variable substitution in OPEN statement 1122
prepared SQL statements
  executing 1003
  host variable substitution 1003
  obtaining information
    DESCRIBE INPUT statement 958
    DESCRIBE OUTPUT statement 962
primary keys
  adding
    ALTER TABLE statement 114
    CREATE TABLE statement 672
  dropping
    ALTER TABLE statement 114
  privileges required 1078
privileges
  databases
    revoking 1175
  indexes
    revoking 1162
  packages
    revoking 1166, 1187
  revoking
    REVOKE statement 1187

problem determination
information available 1354
tutorials 1354
procedure designator syntax element 20
procedures
authorization for creating
CREATE PROCEDURE (external) statement 604
CREATE PROCEDURE (SQL) statement 626
CALL statement 276
CREATE PROCEDURE statement 603
creating 604, 626
PROGRAM option for DB2 for z/OS compatibility
DROP statement 969
PROGRAM TYPE
CREATE FUNCTION (external scalar) statement 436
CREATE FUNCTION (external table) statement 464
PUBLIC AT ALL LOCATIONS 1078

Q

question mark
EXECUTE parameter marker 1003

R

read-only cursors
ambiguous 928
read-only views
creating 879
REAL SQL data type
CREATE TABLE statement 672
records
locks on row data 1091
References
labels 18
SQL condition names 18
SQL cursor names 19
SQL statement names 19
referential constraints
adding comments to catalog 289
REFRESH TABLE statement 1136
RELEASE (connection) statement 1140
RELEASE SAVEPOINT statement 1142
remote access
CONNECT statement 329
successful connections 329
unsuccessful connections 329
RENAME statement 1143
RENAME STOGROUP statement
details 1145
RENAME TABLESPACE statement 1146
REPEAT statement
details 1147
RESIGNAL statement 1149
RESTRICT delete rule 672
result sets
returning
SQL procedures 312
RESULTSTATUS parameter 1034
return codes
embedded statements 10, 12
executable SQL statements 10, 12
RETURN statement
details 1152
returning result sets
SQL procedures 312
REVOKE statement
database authorities 1154
exemptions 1158
global variable privileges 1160
index privileges 1162
module privileges 1164
nickname privileges 1187
package privileges 1166
roles 1169
routine privileges 1171
schema privileges 1175
security labels 1177
sequence privileges 1179
server privileges 1181
SETSESSIONUSER privilege 1183
table privileges 1187
table space privileges 1185
view privileges 1187
workload privileges 1192
XSR object privileges 1194
REXX language
END DECLARE SECTION prohibition 1002
ROLLBACK statement
details 1195
ROLLBACK TO SAVEPOINT statement 1195
row data types
CREATE TYPE (cursor) statement 817
row fullselect
UPDATE statement 1321
rows
assigning values to host variables
SELECT INTO statement 1202
VALUES INTO statement 1338
cursors
effect of closing on FETCH statement 287
FETCH statement 1122
location in result tables 928
deleting
DELETE statement 947
FETCH request 928
granting privileges 1078
index keys with UNIQUE clause 545
indexes 545
inserting
INSERT statement 1091
locks
effect on cursor of WITH HOLD 928
INSERT statement 1091
restrictions leading to failure 1091
updating
column values by using UPDATE statement 1321

S

SAVEPOINT statement 1198
savepoints
releasing 1142
ROLLBACK statement with TO SAVEPOINT clause 1195
schemas
adding comments to catalog 289
CREATE SCHEMA statement 637
implicit
granting authority 1039
revoking authority 1154
scope
adding with ALTER TABLE statement 114
adding with ALTER VIEW statement 227

scope *(continued)*
CREATE VIEW statement 879
defining with added columns 114
defining with CREATE TABLE statement 672
search conditions
DELETE statement 947
UPDATE statement 1321
SECADM (security administrator) authority
granting 1039
revoking 1154
security
CONNECT statement 329
security labels (LBAC)
ALTER SECURITY LABEL COMPONENT statement 86
CREATE SECURITY LABEL COMPONENT statement 640
CREATE SECURITY LABEL statement 643
GRANT (security label) statement 1066
policies
ALTER SECURITY POLICY statement 89
CREATE SECURITY POLICY statement 645
REVOKE (security label) statement 1177
SELECT INTO statement
details 1202
SELECT statement
cursors 928
details 1201
evaluating for result table of OPEN statement cursor 1122
select-statement SQL statement construct
definition 11, 12
dynamic invocation 11
invoking 10
static invocation 11
sequences
DROP statement 969
servers
granting privileges 1072
SET COMPILATION ENVIRONMENT statement 1206
SET CONNECTION statement 1207
SET CONSTRAINTS statement 1257
SET CURRENT DECFLOAT ROUNDING MODE statement 1209
SET CURRENT DEFAULT TRANSFORM GROUP statement 1211
SET CURRENT DEGREE statement 1212
SET CURRENT EXPLAIN MODE statement 1214
SET CURRENT EXPLAIN SNAPSHOT statement 1217
SET CURRENT FEDERATED ASYNCHRONY statement 1219
SET CURRENT FUNCTION PATH statement 1278
SET CURRENT IMPLICIT XMLPARSE OPTION statement 1221
SET CURRENT ISOLATION statement 1222
SET CURRENT LOCALE LC_MESSAGES statement 1223
SET CURRENT LOCALE LC_TIME statement 1225
SET CURRENT LOCK TIMEOUT statement 1227
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION statement 1229
SET CURRENT MDC ROLLOUT MODE statement 1231
SET CURRENT OPTIMIZATION PROFILE statement 1233
SET CURRENT PACKAGE PATH statement 1236
SET CURRENT PACKAGESET statement 1240
SET CURRENT PATH statement 1278
SET CURRENT QUERY OPTIMIZATION statement
details 1242
SET CURRENT REFRESH AGE statement 1245
SET CURRENT SQL_CCFLAGS statement 1247
SET CURRENT SQLID statement 1281
SET CURRENT TEMPORAL BUSINESS_TIME statement 1249
SET CURRENT TEMPORAL SYSTEM_TIME statement 1251
SET ENCRYPTION PASSWORD statement
details 1253
SET EVENT MONITOR STATE statement 1255
set integrity pending state
SET INTEGRITY statement 1257
SET INTEGRITY statement
details 1257
SET NULL delete rule 672
SET PASSTHRU statement
details 1276
independence from COMMIT statement 300
independence from ROLLBACK statement 1195
SET PATH statement 1278
SET ROLE statement 1280
SET SCHEMA statement 1281
SET SERVER OPTION statement
details 1283
independence from COMMIT statement 300
independence from ROLLBACK statement 1195
SET SESSION AUTHORIZATION statement 1285
SET USAGE LIST STATE statement 1288
SET variable statement 1291
SETSESSIONUSER privilege
GRANT (SETSESSIONUSER privilege) statement 1074
required for SET SESSION AUTHORIZATION statement 1285
REVOKE (SETSESSIONUSER privilege) statement 1183
SHARE MODE connection 329
SIGNAL statement 1303
single-precision floating-point data type 672
SMALLINT data type
static SQL 672
SQL
objects
deleting 969
parameters 17
return codes 10
variables
compound SQL (compiled) statement 312
compound SQL (inlined) statement 303
references 17
SQL comments
bracketed 13
simple 13
SQL condition names
references 18
SQL cursor names
references 19
SQL procedures
CASE statement 284
compiled compound statement 312
compound SQL (inlined) statement 303
condition handlers
declaring 312
DECLARE statement 303, 312
FOR statement 1030
GET DIAGNOSTICS statement 1034
GOTO statement 1037
IF statement 1087
ITERATE statement 1102
LEAVE statement 1104
LOOP statement 1108
REPEAT statement 1147
RETURN statement 1152

SQL procedures *(continued)*
SIGNAL statement 1303
variables 303, 312
WHILE statement 1344
SQL return codes 12
SQL statement names
references 19
SQL statements
ALLOCATE CURSOR 24
ALTER AUDIT POLICY 26
ALTER BUFFERPOOL 29
ALTER DATABASE 36
ALTER DATABASE PARTITION GROUP 32
ALTER EVENT MONITOR 41
ALTER FUNCTION 46
ALTER HISTOGRAM TEMPLATE 50
ALTER INDEX 52
ALTER MASK 53
ALTER METHOD 54
ALTER MODULE 56
ALTER NICKNAME 64
ALTER NODEGROUP (see SQL statements, ALTER DATABASE PARTITION GROUP) 32
ALTER PACKAGE 73
ALTER PERMISSION 76
ALTER PROCEDURE (external) 77
ALTER PROCEDURE (sourced) 80
ALTER PROCEDURE (SQL) 82
ALTER SCHEMA 84
ALTER SECURITY LABEL COMPONENT 86
ALTER SECURITY POLICY 89
ALTER SEQUENCE 93
ALTER SERVER 97
ALTER SERVICE CLASS 100
ALTER STOGROUP 110
ALTER TABLE 114
ALTER TABLESPACE 179
ALTER THRESHOLD 194
ALTER TRIGGER 207
ALTER TRUSTED CONTEXT 208
ALTER TYPE (structured) 216
ALTER USAGE LIST 223
ALTER USER MAPPING 225
ALTER VIEW 227
ALTER WORK ACTION SET 229
ALTER WORK CLASS SET 243
ALTER WORKLOAD 249
ALTER WRAPPER 265
ALTER XSROBJECT 267
ASSOCIATE LOCATORS 268
AUDIT 270
BEGIN DECLARE SECTION 274
CALL 276
CLOSE 287
COMMENT 289
COMMIT 300
compound (embedded) 308
compound SQL 302
CONNECT
type 1 329
type 2 336
CONTINUE 1341
control 17
CREATE ALIAS 343
CREATE AUDIT POLICY 347
CREATE BUFFERPOOL 350
CREATE DATABASE PARTITION GROUP 354
SQL statements *(continued)*
CREATE EVENT MONITOR 356
CREATE EVENT MONITOR (activities) 376
CREATE EVENT MONITOR (change history) 387
CREATE EVENT MONITOR (package cache) 400
CREATE EVENT MONITOR (statistics) 407
CREATE EVENT MONITOR (threshold violations) 419
CREATE FUNCTION
external scalar 436
external table 464
OLE DB external table 484
overview 435
sourced 495
SQL row 509
SQL scalar 509
SQL table 509
template 495
CREATE FUNCTION MAPPING 526
CREATE GLOBAL TEMPORARY TABLE 530
CREATE HISTOGRAM TEMPLATE 543
CREATE INDEX 545
CREATE INDEX EXTENSION 566
CREATE MASK 572
CREATE METHOD 578
CREATE MODULE 584
CREATE NICKNAME 586
CREATE NODEGROUP (see SQL statements, CREATE DATABASE PARTITION GROUP) 354
CREATE PERMISSION 599
CREATE PROCEDURE
external 604
overview 603
sourced 620
SQL 626
CREATE ROLE 636
CREATE SCHEMA 637
CREATE SECURITY LABEL 643
CREATE SECURITY LABEL COMPONENT 640
CREATE SECURITY POLICY 645
CREATE SEQUENCE 647
CREATE SERVER 664
CREATE SERVICE CLASS 653
CREATE STOGROUP 668
CREATE TABLE 672
CREATE TABLESPACE 752
CREATE THRESHOLD 767
CREATE TRANSFORM 784
CREATE TRIGGER 788
CREATE TRUSTED CONTEXT 803
CREATE TYPE 810
array 811
distinct 820
row 828
structured 833
CREATE TYPE MAPPING 856
CREATE USAGE LIST 863
CREATE USER MAPPING 867
CREATE VARIABLE 869
CREATE VIEW 879
CREATE WORK ACTION SET 894
CREATE WORK CLASS SET 903
CREATE WORKLOAD 908
CREATE WRAPPER 926
DECLARE CURSOR 928
DECLARE GLOBAL TEMPORARY TABLE 934
DELETE 947
DESCRIBE 957

SQL statements *(continued)*
DESCRIBE INPUT 958
DESCRIBE OUTPUT 962
DISCONNECT 966
DROP 969
DROP TRANSFORM 969
embedded 10
END DECLARE SECTION 1002
EXECUTE 1003
EXECUTE IMMEDIATE 1011
EXPLAIN 1014
FETCH 1019
FLUSH BUFFERPOOLS 1023
FLUSH EVENT MONITOR 1024
FLUSH FEDERATED CACHE 1025
FLUSH OPTIMIZATION PROFILE CACHE 1027
FLUSH PACKAGE CACHE 1029
FREE LOCATOR 1033
GRANT
database authorities 1039
exemption 1044
global variable privileges 1047
index privileges 1049
module privileges 1051
nickname privileges 1078
package privileges 1053
role 1056
routine privileges 1059
schema privileges 1063
security label 1066
sequence privileges 1069
server privileges 1072
SETSESSIONUSER privilege 1074
table privileges 1078
table space privileges 1076
view privileges 1078
workload privileges 1084
XSR object privileges 1086
help
displaying 1350
INCLUDE 1089
INSERT 1091
interactive entry 10, 12
invoking 10
LOCK TABLE 1106
MERGE 1110
OPEN 1122
overview 1
PIPE 1128
PREPARE 1130
REFRESH TABLE 1136
RELEASE (connection) 1140
RELEASE SAVEPOINT 1142
RENAME 1143
RENAME STOGROUP 1145
RENAME TABLESPACE 1146
RESIGNAL 1149
REVOKE
database authorities 1154
exemption 1158
global variable privileges 1160
index privileges 1162
nickname privileges 1187
package privileges 1166
role 1169
routine privileges 1171
schema privileges 1175
SQL statements *(continued)*
REVOKE *(continued)*
security label 1177
sequence privileges 1179
server privileges 1181
SETSESSIONUSER privilege 1183
table privileges 1187
table space privileges 1185
view privileges 1187
workload privileges 1192
XSR object privileges 1194
ROLLBACK 1195
ROLLBACK TO SAVEPOINT 1195
SAVEPOINT 1198
SELECT 1201
SELECT INTO 1202
SET COMPILATION ENVIRONMENT 1206
SET CONNECTION 1207
SET CONSTRAINTS 1257
SET CURRENT DECFLOAT ROUNDING MODE 1209
SET CURRENT DEFAULT TRANSFORM GROUP 1211
SET CURRENT DEGREE 1212
SET CURRENT EXPLAIN MODE 1214
SET CURRENT EXPLAIN SNAPSHOT 1217
SET CURRENT FEDERATED ASYNCHRONY 1219
SET CURRENT FUNCTION PATH 1278
SET CURRENT IMPLICIT XMLPARSE OPTION 1221
SET CURRENT ISOLATION 1222
SET CURRENT LOCALE LC_MESSAGES 1223
SET CURRENT LOCK TIMEOUT 1227
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 1229
SET CURRENT MDC ROLLOUT MODE 1231
SET CURRENT OPTIMIZATION PROFILE 1233
SET CURRENT PACKAGE PATH 1236
SET CURRENT PACKAGESET 1240
SET CURRENT PATH 1278
SET CURRENT QUERY OPTIMIZATION 1242
SET CURRENT REFRESH AGE 1245
SET CURRENT SQL_CCFLAGS 1247
SET CURRENT TEMPORAL BUSINESS_TIME 1249
SET CURRENT TEMPORAL SYSTEM_TIME 1251
SET ENCRYPTION PASSWORD 1253
SET EVENT MONITOR STATE 1255
SET INTEGRITY 1257
SET PASSTHRU 1276
SET PATH 1278
SET ROLE 1280
SET SCHEMA 1281
SET SERVER OPTION 1283
SET SESSION AUTHORIZATION 1285
SET USAGE LIST STATE 1288
SET variable 1291
strings
creating 1011
PREPARE statement 1130
TRANSFER OWNERSHIP 1306
TRUNCATE 1318
UPDATE 1321
VALUES 1337
VALUES INTO 1338
WHENEVER 1341
WITH HOLD cursor attribute 928
SQL92 standard
dynamic SQL 1281
SQLCA
overview 10

SQLCA *(continued)*
UPDATE statement 1321
SQLCA clause of INCLUDE statement 1089
SQLCA structure
overview 12
SQLCODE
description 12
details 10
SQLDA
FETCH statement 1019
host variable details 1122
OPEN statement 1122
required variables for DESCRIBE INPUT statement 958
required variables for DESCRIBE OUTPUT statement 962
SQLERROR clause of WHENEVER statement 1341
SQLSTATE
description 12
details 10
SQLWARNING clause of WHENEVER statement 1341
standards
setting rules for dynamic SQL 1281
start key values 566
statements
LEAVE 1104
static SQL
DECLARE CURSOR statement 10, 11
FETCH statement 10
invoking 10, 11
OPEN statement 10
select 11
select-statement 10
statements 10, 11
STAY RESIDENT
CREATE FUNCTION (external scalar) statement 436
CREATE FUNCTION (external table) statement 464
CREATE PROCEDURE statement 604, 626
stop key values 566
storage structures
ALTER BUFFERPOOL statement 29
ALTER TABLESPACE statement 179
CREATE BUFFERPOOL statement 350
CREATE TABLESPACE statement 752
structured types
CREATE TRANSFORM statement 784
DROP statement 969
summary tables
overview 672
synonyms
CREATE ALIAS statement 343
DROP ALIAS statement 969
syntax diagrams
reading viii
system-managed space (SMS)
table spaces
CREATE TABLESPACE statement 752

T

table spaces
adding
comments to catalog 289
buffer pools 350
creating
CREATE TABLESPACE statement 752
deleting using DROP statement 969
dropping
DROP statement 969
table spaces *(continued)*
granting privileges 1076
identifying
CREATE TABLE statement 672
indexes
CREATE TABLE statement 672
page sizes 752
renaming 1146
revoking privileges 1185
tables
adding columns 114
adding comments to catalog 289
aliases 343, 969
altering
ALTER TABLE statement 114
authorization for creating 672
creating
granting authority 1039
SQL statement instructions 672
deleting using DROP statement 969
exception 1257
generated columns 114
granting privileges 1078
indexes 545
inserting rows 1091
joining
CREATE TABLE statement 672
names
ALTER TABLE statement 114
CREATE TABLE statement 672
LOCK TABLE statement 1106
renaming 1143
restricting shared access 1106
revoking privileges 1187
schemas 637
temporary
OPEN statement 1122
typed
triggers 788
updating by row and column 1321
temporary tables
OPEN statement 1122
termination
units of work 300, 1195
terms and conditions
publications 1355
TIME data types
CREATE TABLE statement 672
TIMESTAMP data type
CREATE TABLE statement 672
TRANSFER OWNERSHIP statement 1306
transform functions
CREATE TRANSFORM 784
transformations
DROP statement 969
triggered SQL statements
SET variable 1291
triggers
adding comments to catalog 289
ALTER TRIGGER statement 207
CREATE TRIGGER statement 788
dropping 969
error messages 788
inoperative 207, 788
INSERT statement 1091
typed tables 788
UPDATE statement 1321

troubleshooting
  online information 1354
  tutorials 1354
TRUNCATE statement
  details 1318
tutorials
  list 1354
  problem determination 1354
  pureXML 1354
  troubleshooting 1354
type 2 indexes 545
typed views
  defining subviews 879

U

UDFs
  CREATE FUNCTION statement
    external scalar 436
    external table 464
    OLE DB external table 484
    overview 435
    sourced 495
    SQL scalar, table, or row 509
    template 495
  DROP statement 969
  REVOKE (database authorities) statement 1154
UDTs
  adding comments to catalog 289
  CREATE TRANSFORM statement 784
  CREATE TYPE (distinct) statement 820
  distinct types
    CREATE TABLE statement 672
  structured types 672
unique constraints
  adding
    ALTER TABLE statement 114
  ALTER TABLE statement 114
  CREATE TABLE statement 672
  dropping with ALTER TABLE 114
unique keys
  ALTER TABLE statement 114
  CREATE TABLE statement 672
units of work
  canceling 1195
  COMMIT statement 300
  destroying prepared statements 1130
  initiation closes cursors 1122
  referring to prepared statements 1130
  ROLLBACK statement 1195
  terminating
    commits 300
    destroys prepared statements 1130
    without saving changes 1195
UPDATE statement
  details 1321
updates
  DB2 Information Center 1351, 1352
  updatable views 879
usage lists
  deleting using DROP statement 969

V

VALUES clause
  loading one row 1091
VALUES clause *(continued)*
  rules for number of values 1091
VALUES INTO statement 1338
VALUES statement 1337
VARCHAR data type
  CREATE TABLE statement 672
VARIANT
  CREATE TYPE (structured) statement 833
views
  adding comments to catalog 289
  aliases 343, 969
  column names 879
  CONTROL privilege 1078
  creating 879
  deletable 879
  deleting using DROP statement 969
  granting privileges 1078
  inoperative 879
  insertable 879
  inserting rows 1091
  names 227
  preventing view definition loss with WITH CHECK OPTION 1321
  read-only 879
  revoking privileges 1187
  schemas 637
  updatable 879
  updating rows by columns 1321
  WITH CHECK OPTION 1321

W

WHENEVER statement
  changing flow of control 10
  details 1341
WHERE clause
  DELETE statement 947
  UPDATE statement 1321
WHILE statement
  details 1344

X

XML
  CREATE INDEX statement 545
XML data
  CREATE INDEX statement 545

Printed in USA

SC27-3886-01

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

SQL Reference Volume 2