Db2 11 for z/OS

Introduction to Db2 for z/OS



SC19-4058-10

Notes

Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

Subsequent editions of this PDF will not be delivered in IBM Publications Center. Always download the latest edition from PDF format manuals for Db2 11 for z/OS (Db2 Home).

2020-07-30 edition

This edition applies to Db2® 11 for z/OS® (product number 5615-DB2), Db2 11 for z/OS Value Unit Edition (product number 5697-P43), and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© Copyright International Business Machines Corporation 2001, 2020.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this information..... ix
Who should read this information..... ix
Db2 Utilities Suite for z/OS..... ix
Terminology and citations..... x
Accessibility features for Db2 11 for z/OS..... xi
How to send your comments..... xi

Chapter 1. What is Db2 for z/OS?..... 1

Chapter 2. Db2 for z/OS and SQL concepts..... 3
Submitting SQL statements to Db2..... 3
Static SQL..... 4
Embedded dynamic SQL..... 5
Deferred embedded SQL..... 5
Interactive SQL..... 5
SQL Call Level Interface and Open Database Connectivity..... 6
Java database connectivity and embedded SQL for Java..... 7
Use of QMF for Workstation..... 7
Db2 database objects overview..... 8
Db2 schemas and schema qualifiers..... 10
Db2 tables..... 11
Db2 keys..... 12
Constraints..... 14
Db2 columns..... 18
Db2 indexes..... 31
Db2 views..... 31
Aliases..... 33
Triggers..... 34
User-defined types..... 35
Routines..... 35
Functions..... 35
Procedures..... 36
Sequences..... 37
Db2 system objects..... 38
Db2 catalog..... 38
Db2 directory..... 39
Active and archive logs..... 40
Bootstrap data set..... 40
Buffer pools..... 41
Data definition control support database..... 41
The resource limit facility..... 41
Work file database..... 42
Application processes and transactions..... 42
Subsystem parameters..... 47
Storage structures..... 47
Db2 databases..... 48
Db2 table spaces..... 50
Db2 index spaces..... 51
Rules for primary and secondary space allocation..... 51
Db2 hash spaces (deprecated)..... 53

Db2 storage groups 54
Accessing Db2 data 55
Ways to select data from columns 56
How a SELECT statement works 58
SQL functions and expressions 59
Ways to filter the number of returned rows 65
Ways to order rows 72
Ways to summarize group values 74
Ways to merge lists of values 75
Ways to specify search conditions 76
Ways to join data from more than one table 77
Subqueries 84
Ways to access Db2 data that is not in a table 85
Modifying Db2 data 85
Insert statements 86
Update statements 87
Merge statements 87
Delete statements 87
Truncate statements 88
Defining Db2 objects 88
Creation of tables 88
Creation of table spaces 96
Creation of indexes 100
Creation of views 117
Creation of large objects 119
Creation of databases 120
Creation of relationships with referential constraints 121
Creation of triggers 125
Creation of user-defined functions 126
Application processes, concurrency, and recovery 127
Locking, commit, and rollback 127
Unit of work 128
Unit of recovery 128
Rolling back work 129
Packages and application plans 130
Character conversion 131
Character sets and code pages 133
Coded character sets and CCSIDS 135
Determining the encoding scheme and CCSID of a string 135
Expanding conversions 140
Contracting conversions 140
Distributed relational databases 140
Connections 141
Distributed unit of work 141
Remote unit of work 144
Distributed data 146
Remote servers 146
Connectivity in distributed environments 147
Distributed data access 147
Support for high availability 156
Db2 data sharing 156
Advantages of Db2 data sharing 157
How Db2 protects data consistency in a data sharing environment 162
How updates are made in a data sharing environment 163
How Db2 writes changed data to disk in a data sharing environment 166
Ways that other tasks are affected by data sharing 167
Ways that availability is affected by data sharing 167
pureXML 168

Chapter 3. Management of Db2 operations..... 171
Common ways to interact with Db2 for z/OS..... 171
The DB2I primary option menu..... 173
Commands for controlling Db2 and related facilities..... 174
Db2 utilities..... 176
Management of data sets..... 181
Authorization and security mechanisms for data access..... 181
How authorization IDs control data access..... 182
How authorization IDs hold privileges and authorities..... 182
Ways to control access to Db2 subsystems..... 184
Ways to control access to data..... 185
Ways to control access to Db2 objects through explicit privileges and authorities..... 186
Row-level and column-level access control..... 188
Use of multilevel security to control access..... 188
Use of views to control access..... 189
Use of grant and revoke privileges to control access..... 189
Backup, recovery, and restart..... 191
Backup and recovery resources and tools..... 193
Db2 restart..... 195
Regular backups and data checks..... 195
Control of database changes and data consistency..... 195
Events in the recovery process..... 197
Optimization of availability during backup and recovery..... 198
Data encoding basics..... 199
Introduction to troubleshooting problems in Db2 for z/OS..... 201

Chapter 4. Db2 performance management..... 207
Initial steps for performance management..... 207
Performance objectives..... 207
Application design for performance..... 208
Origin of performance problems..... 208
Tools for performance analysis..... 209
Ways to move data efficiently through the system..... 210
The role of buffer pools in caching data..... 210
The effect of data compression on performance..... 212
How data organization can affect performance..... 212
Ways to improve performance for multiple users..... 215
Improved performance through the use of locks..... 215
Improved performance through concurrency control..... 218
Ways to improve query performance..... 220
Tools for improving query performance..... 221
Query and application performance analysis..... 222
Using EXPLAIN to understand the access path..... 225
Hash access paths..... 228

Chapter 5. Programming for Db2 for z/OS..... 229
Tools and IDEs for developing Db2 applications..... 233
Programming languages and methods for developing application programs that access Db2 for z/OS..... 235
Performance information for SQL application programming..... 236
Preparation process for an application program..... 238
Application programming default values..... 240
Static SQL applications..... 241
Declaration of table and view definitions..... 242
Data access with host variables..... 242
Data access with host variable arrays..... 244

Data access with host structures..... 244
Row retrieval with a cursor..... 245
Ways to check the execution of SQL statements..... 247
Dynamic SQL applications..... 247
Use of ODBC to execute dynamic SQL..... 248
Use of Java to execute static and dynamic SQL..... 250
SQLJ support..... 251
JDBC support..... 251
Use of an application program as a stored procedure..... 252
Languages used to create stored procedures..... 253
Stored procedure processing..... 253
Use of the SQL procedural language to create a stored procedure..... 256
Use of development tools to create a stored procedure..... 256
Setup of the stored procedure environment..... 257
Preparation of a stored procedure..... 257
How applications can call stored procedures..... 257
Web application environment..... 258
Components of web-based applications..... 258
Architectural characteristics of web-based applications..... 259
Benefits of Db2 for z/OS as a server..... 261
Web-based applications and WebSphere Studio Application Developer..... 261
Processing XML data with Db2 pureXML..... 263

Chapter 6. Db2 database design..... 267
Db2 objects and their relationships..... 267
Logical database design using entity-relationship modeling..... 268
Logical database design with Unified Modeling Language..... 277
Physical database design..... 279
Row design for effective space use..... 283
XML and Db2..... 284

Chapter 7. Db2 for z/OS architecture..... 287
z/Architecture and the z/OS operating system..... 287
Db2 in the z/OS environment..... 289
Db2 internal resource lock manager..... 290
Db2 and the z/OS Security Server..... 290
Db2 and DFSMS..... 291
Db2 attachment facilities..... 291
CICS attachment facility..... 292
IMS attachment facility..... 293
TSO attachment facility..... 294
Call attachment facility..... 294
Resource Recovery Services attachment facility..... 294
Distributed data facility..... 295
Db2 in a Parallel Sysplex environment..... 295

Chapter 8. An overview of Db2 for z/OS and related products..... 297
Scenarios for using Db2..... 297
Availability and scalability for large businesses..... 297
Critical business information for decision makers..... 299
Data distribution and Web access..... 300
Db2 data servers and environments..... 301
Enterprise servers..... 302
Db2 distributed editions..... 302
Db2 on smaller-scale servers..... 302
Personal, mobile, and pervasive environments..... 302
Multiple transaction and application environments..... 302

Db2 and network communication ... 303
Clients supported by Db2 data servers ... 303
Sources of data ... 304
Tools for managing Db2 for z/OS ... 305
IBM Data Server Manager ... 305
IBM Data Studio ... 307
Db2 Administration Tool for z/OS ... 308
The Db2 command line processor ... 308
Tools and IDEs for developing Db2 applications ... 308
Middleware components ... 310
IBM Rational Portfolio Manager ... 310
Db2 Connect ... 310
WebSphere Application Server ... 311
WebSphere Studio ... 311
WebSphere Host Integration ... 312
Federated database support through WebSphere Information Integrator ... 312
Data replication through InfoSphere Replication Server ... 312
WebSphere DataStage ... 313
WebSphere QualityStage ... 313
Client application programming interfaces ... 313
Open standards ... 315
Db2 and the web ... 315
SOA, XML, and web services ... 316

Chapter 9. Db2 sample tables ... 319
Activity table (DSN8B10.ACT) ... 319
Department table (DSN8B10.DEPT) ... 320
Employee table (DSN8B10.EMP) ... 322
Employee photo and resume table (DSN8B10.EMP_PHOTO_RESUME) ... 326
Project table (DSN8B10.PROJ) ... 327
Project activity table (DSN8B10.PROJACT) ... 328
Employee-to-project activity table (DSN8B10.EMPPROJACT) ... 329
Unicode sample table (DSN8B10.DEMO_UNICODE) ... 330
Relationships among the sample tables ... 331
Views on the sample tables ... 332
Storage of sample application tables ... 336
Storage group for sample application data ... 337
Databases for sample application data ... 337
Table spaces for sample application data ... 337
SYSDUMMY*x* tables ... 339

Chapter 10. Db2 basics tutorial: Working with Db2 for z/OS interactively ... 341
Module 1: Querying and modifying data ... 342
Lesson 1.1: Querying data interactively ... 342
Lesson 1.2: Creating a table ... 348
Lesson 1.3: Modifying data interactively ... 350
Module 2: Running Db2 utilities ... 353
Lesson 2.1: Running the UNLOAD utility from DB2I ... 354
Lesson 2.2: Running the LOAD utility from DB2I ... 361
Lesson 2.3: Running the COPY utility from a JCL job ... 364

Chapter 11. New-user information roadmap ... 369

Chapter 12. Db2 11 for z/OS information roadmap ... 375

Chapter 13. Application Developer information roadmap ... 381

Information resources for Db2 11 for z/OS and related products..... 385

Notices.....387
Programming interface information.....388
Trademarks.....388
Terms and conditions for product documentation.....389
Privacy policy considerations.....389

Glossary.....391

Index..... 393

About this information

This information provides a comprehensive introduction to IBM® Db2 for z/OS. It explains the basic concepts that are associated with relational database management systems in general, and with Db2 for z/OS in particular.

After reading this information, you will understand basic concepts about Db2.

Throughout this information, "Db2" means "Db2 11 for z/OS". References to other Db2 products use complete names or specific abbreviations.

Important: To find the most up to date content for Db2 11 for z/OS, always use IBM Knowledge Center or download the latest PDF file from PDF format manuals for Db2 11 for z/OS (Db2 Home).

This information assumes that Db2 11 is running in new-function mode, and that your application is running with the application compatibility value of 'V11R1'.

Availability of new function in Db2 11

The behavior of data definition statements such as CREATE, ALTER, and DROP, which embed data manipulation SQL statements that contain new capabilities, depends on the application compatibility value that is in effect for the application. An application compatibility value of 'V11R1' must be in effect for applications to use new capability in embedded statements such as SELECT, INSERT, UPDATE, DELETE, MERGE, CALL, and SET *assignment-statement*. Otherwise, an application compatibility value of 'V10R1' can be used for data definition statements.

Generally, new SQL capabilities, including changes to existing language elements, functions, data manipulation statements, and limits, are available only in new-function mode with applications set to an application compatibility value of 'V11R1'.

Optimization and virtual storage enhancements are available in conversion mode unless stated otherwise.

SQL statements can continue to run with the same expected behavior as in DB2® 10 new-function mode with an application compatibility value of 'V10R1'.

Who should read this information

If you are new to Db2 for z/OS, this information is for you.

Perhaps you have worked with other Db2 family products, such as Db2 for Linux®, UNIX, and Windows, DB2 for i, or DB2 for VSE & VM. Perhaps you have worked on non-IBM database management systems (DBMSs) or on the IBM hierarchic DBMS, which is called Information Management System (IMS). Perhaps you have never worked with DBMSs, but you want to work with this product, which many companies use for mission-critical data and application programs. Regardless of your background, if you want to learn about Db2 for z/OS, this information can help you.

If you will be working with Db2 for z/OS and already know what specific job you will have, begin by reading the first three chapters. Then, you can consider what your role will be when you choose to read all or only a subset of the remaining chapters. For example, assume that you know you will be a database administrator (DBA) for an organization that has some distributed applications and is beginning to plan for on demand business. In this case you would probably want to read the chapters about designing objects and data, implementing your database design, Db2 and the Web, and accessing distributed data.

This information is written with the assumption that most readers are data processing professionals.

Db2 Utilities Suite for z/OS

Important: In Db2 11, the Db2 Utilities Suite for z/OS is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

© Copyright IBM Corp. 2001, 2020

Db2 11 utilities can use the DFSORT program regardless of whether you purchased a license for DFSORT on your system. For more information, see the following informational APARs:

- II14047
- II14213
- II13495

Db2 utilities can use IBM Db2 Sort for z/OS (5655-W42) as an alternative to DFSORT for utility SORT and MERGE functions. Use of Db2 Sort for z/OS requires the purchase of a Db2 Sort for z/OS license. For more information about Db2 Sort for z/OS, see Db2 Sort for z/OS.

Related concepts

Db2 utilities packaging (Db2 Utilities)

Terminology and citations

When referring to a Db2 product other than Db2 for z/OS, this information uses the product's full name to avoid ambiguity.

About the Db2 brand change: IBM has re-branded DB2 to Db2, and Db2 for z/OS is now the name of the offering previously know as "DB2 for z/OS". During and after implementation of this change, you might sometimes still see references to the original names, such as "DB2 for z/OS" and "DB2", in different IBM web pages and documents. However, assume that these refer to the same "Db2 for z/OS" product when the PID, Entitlement Entity, version, modification, and release information match. For more information, see Revised naming for IBM Db2 family products on IBM z/OS platform.

The following terms are used as indicated:

Db2
Represents either the Db2 licensed program or a particular Db2 subsystem.

Tivoli® OMEGAMON® XE for Db2 Performance Expert on z/OS
Refers to any of the following products:

- IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS
- IBM Db2 Performance Monitor on z/OS
- IBM Db2 Performance Expert for Multiplatforms and Workgroups
- IBM Db2 Buffer Pool Analyzer for z/OS

C, C++, and C language
Represent the C or C++ programming language.

CICS®
Represents CICS Transaction Server for z/OS.

IMS
Represents the IMS Database Manager or IMS Transaction Manager.

MVS™
Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®
Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for Db2 11 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including Db2 11 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: The IBM Knowledge Center (which includes information for Db2 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the Db2 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other Db2 for z/OS documentation.

Send your comments by email to db2zinfo@us.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).

Chapter 1. What is Db2 for z/OS?

Db2 for z/OS is a relational database management system that runs on the mainframe.

A *relational database* is a database in which all of the data is logically contained in tables. These databases are organized according to the relational model. In a relational database, referential integrity ensures data integrity by enforcing rules with referential constraints, check constraints, and triggers. You can rely on constraints and triggers to ensure the integrity and validity of your data, rather than relying on individual applications to do that work.

Db2 is software that you can use to manage relational databases. IBM offers a family of Db2 products that run on a range of operating systems, including Linux, UNIX, Windows, IBM i, VSE, VM, and z/OS.

z/OS is the main operating system for IBM's most robust hardware platform, IBM Z. Db2 for z/OS is the enterprise data server for IBM Z. It manages core business data across an enterprise and supports key business applications. Db2 for z/OS supports thousands of customers and millions of users. It remains continuously available, scalable, and highly secure.

With Db2 for z/OS and other Db2 family products, you can define and manipulate your data by using structured query language (SQL). SQL is the standard language for accessing data in relational databases.

This information might sometimes also refer to Db2 11 for z/OS as "Db2" or "Version 11."

About the Db2 brand change: IBM has re-branded DB2 to Db2, and Db2 for z/OS is now the name of the offering previously know as "DB2 for z/OS". During and after implementation of this change, you might sometimes still see references to the original names, such as "DB2 for z/OS" and "DB2", in different IBM web pages and documents. However, assume that these refer to the same "Db2 for z/OS" product when the PID, Entitlement Entity, version, modification, and release information match. For more information, see Revised naming for IBM Db2 family products on IBM z/OS platform.

Db2 for z/OS News from the Lab blog: See the Db2 for z/OS News from the Lab blog for the latest news about new capabilities and enhancements in Db2 for z/OS continuous delivery, from the IBM experts who design, build, test, and support Db2

Subscribe today!

Related concepts

Scenarios for using Db2
Scenarios can illustrate how some organizations might successfully use Db2.

SQL: The language of Db2 (Db2 SQL)

Related information

IBM Db2 database software

IBM Mainframe: Make the Extraordinary Possible

zIQ by IBM System z

© Copyright IBM Corp. 2001, 2020

Chapter 2. Db2 for z/OS and SQL concepts

Many structures and processes are associated with a relational database. The structures are the key components of a Db2 database system, and the processes are the interactions that occur when applications access the database system.

In a relational database, data is perceived to exist in one or more *tables*. Each table contains a specific number of *columns* and a number of unordered *rows*. Each column in a table is related in some way to the other columns. Thinking of the data as a collection of tables gives you an easy way to visualize the data that is stored in a Db2 database.

Tables are at the core of a Db2 database. However, a Db2 database involves more than just a collection of tables; a Db2 database also involves other objects, such as views and indexes, and larger data containers, such as table spaces.

With Db2 for z/OS and the other Db2 products, you can define and manipulate your data by using structured query language (SQL). SQL is the standard language for accessing data in relational databases.

Related concepts

SQL: The language of Db2 (Db2 SQL)

Submitting SQL statements to Db2

You can use several different methods to send SQL statements to Db2 in several ways.

You can issue SQL statements from interactively from a keyboard at a terminal or through a terminal emulator.

Another way to issue SQL statements is through application programs. The programs can contain SQL statements that are statically embedded in the application. Alternatively, application programs can create their SQL statements dynamically, for example, in response to information that a user provides by filling in a form.

The method of preparing an SQL statement for execution and the persistence of its operational form distinguish static SQL from dynamic SQL.

Interactive SQL

Interactive SQL refers to SQL statements that you submit to Db2 by using SPUFI (SQL processor using file input), the command line processor, or by using query tools, such as Db2 Query Management Facility (QMF).

Related information:

"Interactive SQL" on page 5
Executing SQL by using SPUFI (Db2 Application programming and SQL)
The Db2 command line processor (Db2 Commands)

Static SQL

The source form of a *static SQL statement* is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

Related information:

"Static SQL" on page 4

Dynamic SQL

You can use dynamic SQL when you do not know the content of an SQL statement when you write a program or before you run it.

You can issue dynamic SQL statements in the following contexts:

- Interactively SQL through SPUFI or tools such as QMF

© Copyright IBM Corp. 2001, 2020

- Embedded dynamic SQL
- Deferred embedded SQL
- Through ODBC functions
- Through JDBC functions.

You can use IBM pureQuery® to add static functionality to dynamic SQL. IBM pureQuery features an intuitive API and enables SQL access to databases or in-memory Java™ objects. You can also use pureQuery Runtime to enable flexible static SQL deployment for Db2.

Related information:

"Dynamic SQL applications" on page 247
Including dynamic SQL in your program (Db2 Application programming and SQL)
Improving dynamic SQL performance (Db2 Performance)

Db2 ODBC

Db2 ODBC (Open Database Connectivity) is an application programming interface (API) that enables C and C++ application programs to access relational databases

This interface offers an alternative to using embedded static SQL and a different way of performing dynamic SQL. Through the interface, an application invokes a C function at execution time to connect to a data source, to dynamically issue SQL statements, and to retrieve data and status information.

Related information:

"SQL Call Level Interface and Open Database Connectivity" on page 6
Introduction to Db2 ODBC (Db2 Programming for ODBC)
ODBC functions (Db2 Programming for ODBC)

Db2 access for Java: SQLJ, JDBC and pureQuery

SQLJ, JDBC, and pureQuery are methods for accessing Db2 data from applications that are written in the Java programming language.

In general, Java applications use SQLJ for static SQL, and they use JDBC for dynamic SQL. IBM pureQuery provides benefits to both static and dynamic SQL.

Related information:

"Use of Java to execute static and dynamic SQL" on page 250
Java application development for IBM data servers (Db2 Application Programming for Java)
Supported drivers for JDBC and SQLJ (Db2 Application Programming for Java)

Related concepts

How SQL statements are invoked (Db2 SQL)

Related tasks

Overview of programming applications that access Db2 for z/OS data (Db2 Application programming and SQL)

Static SQL

The source form of a *static SQL statement* is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

Introductory concepts

"Static SQL applications" on page 241

Static SQL statements in a source program must be processed before the program is compiled. This processing can be accomplished through the Db2 precompiler or the Db2 coprocessor. The Db2 precompiler or the coprocessor checks the syntax of the SQL statements, turns them into host language comments, and generates host language statements to invoke Db2.

The preparation of an SQL application program includes precompilation, the preparation of its static SQL statements, and compilation of the modified source program.

Related concepts

How SQL statements are invoked (Db2 SQL)

Related tasks

Overview of programming applications that access Db2 for z/OS data (Db2 Application programming and SQL)

Embedded dynamic SQL

Programs that contain *embedded dynamic SQL* statements must be precompiled like those that contain static SQL, but unlike static SQL, the dynamic statements are constructed and prepared at run time.

The source form of a dynamic statement is a character string that is passed to Db2 by the program using the static SQL PREPARE or EXECUTE IMMEDIATE statement. A statement that is prepared using the PREPARE statement can be referenced in a DECLARE CURSOR, DESCRIBE, or EXECUTE statement. Whether the operational form of the statement is persistent depends on whether dynamic statement caching is enabled.

SQL statements embedded in a REXX application are dynamic SQL statements. SQL statements submitted to an interactive SQL facility and to the CALL Level Interface (CLI) are also dynamic SQL.

Related concepts

Embedding a statement in an application program (Db2 SQL)

Dynamic preparation and execution (Db2 SQL)

Dynamic SQL applications

With dynamic SQL, Db2 prepares and executes the SQL statements within a program while the program is running. Dynamic SQL is a good choice when you do not know the format of an SQL statement before you write or run a program.

Related tasks

Including dynamic SQL in your program (Db2 Application programming and SQL)

Deferred embedded SQL

A *deferred embedded* SQL statement is neither fully static nor fully dynamic.

Like a static statement, it is embedded within an application, but like a dynamic statement, it is prepared during the execution of the application. Although prepared at run time, a deferred embedded SQL statement is processed with bind-time rules such that the authorization ID and qualifier determined at bind time for the plan or package owner are used.

Interactive SQL

Interactive SQL refers to SQL statements that you submit to Db2 by using SPUFI (SQL processor using file input), the command line processor, or by using a query tool, such as QMF for Workstation.

The easiest and most efficient way to run SQL is to use a query tool. QMF for Workstation is a popular query tool that lets you enter and run your SQL statements easily. This topic acquaints you with using QMF for Workstation to create and run SQL statements. QMF for Workstation simplifies access to Db2 from a workstation. In fact, was QMF for Workstation built for Db2.

Although this topic focuses on QMF for Workstation, other options are available. You can use QMF for WebSphere® to enter and run SQL statements from your web browser or useQMF for TSO/CICS to enter and run SQL statements from TSO or CICS. In addition, you can enter and run SQL statements at a TSO terminal by using the SPUFI (SQL processor using file input) facility. SPUFI prepares and executes these statements dynamically. All of these tools prepare and dynamically execute the SQL statements.

The Db2 Query Management Facility (QMF) family of technologies establish pervasive production and sharing of business intelligence for information-oriented tasks in the organization. QMF offers many strengths, including the following:

- Support for functionality in the Db2 database, including long names, Unicode, and SQL enhancements
- Drag-and-drop capability for building OLAP analytics, SQL queries, pivot tables, and other business analysis and reports
- Executive dashboards and data visual solutions that offer visually rich, interactive functionality and interfaces for data analysis
- Support for QMF for WebSphere, a tool that turns any web browser into a zero-maintenance, thin client for visual on demand access to enterprise Db2 data
- Re-engineered cross-platform development environment
- New security model for access control and personalization

The visual solutions previously provided by QMF Vision are now included in the core QMF technology.

In addition to QMF for Workstation, which this topic describes, the QMF family includes the following editions:

- QMF Enterprise Edition provides the entire QMF family of technologies, enabling enterprise-wide business information across user and database operating systems. This edition consists of:
  - QMF for TSO/CICS
  - QMF High Performance Option
  - QMF for Workstation
  - QMF for WebSphere
  - DataQuant for Workstation
  - DataQuant for WebSphere
- QMF Classic Edition supports users who work with traditional mainframe terminals and emulators (including WebSphere Host On Demand) to access Db2 databases. This edition consists of QMF for TSO/CICS.

Related concepts

Interactive invocation (Db2 SQL)

The Db2 command line processor (Db2 Commands)

"Use of QMF for Workstation" on page 7

QMF for Workstation is a tool that helps you build and manage powerful queries without requiring previous experience with SQL.

Related tasks

Executing SQL by using SPUFI (Db2 Application programming and SQL)

SQL Call Level Interface and Open Database Connectivity

The Db2 Call Level Interface (CLI) is an application programming interface in which functions are provided to application programs to process dynamic SQL statements.

Db2 CLI allows users to access SQL functions directly through a call interface. CLI programs can also be compiled using an Open Database Connectivity (ODBC) Software Developer's Kit, available from Microsoft or other vendors, enabling access to ODBC data sources. Unlike using embedded SQL, no precompilation is required. Applications developed using this interface can be executed on a variety of databases without being compiled against each of databases. Through the interface, applications use procedure calls at execution time to connect to databases, to issue SQL statements, and to get returned data and status information.

Related concepts

Introduction to Db2 ODBC (Db2 Programming for ODBC)

Conceptual view of a Db2 ODBC application (Db2 Programming for ODBC)

Related information

ODBC functions (Db2 Programming for ODBC)

Java database connectivity and embedded SQL for Java

Db2 provides two standards-based Java programming APIs: Java Database Connectivity (JDBC) and embedded SQL for Java (SQL/OLB or SQLJ). Both can be used to create Java applications and applets that access Db2.

Static SQL cannot be used by JDBC. SQLJ applications use JDBC as a foundation for such tasks as connecting to databases and handling SQL errors, but can contain embedded static SQL statements in the SQLJ source files. An SQLJ file has to be translated with the SQLJ translator before the resulting Java source code can be compiled.

Related concepts

Java application development for IBM data servers (Db2 Application Programming for Java)

Supported drivers for JDBC and SQLJ (Db2 Application Programming for Java)

Use of QMF for Workstation

QMF for Workstation is a tool that helps you build and manage powerful queries without requiring previous experience with SQL.

With the query-related features of Db2 Query Management Facility (QMF) and QMF for Workstation in particular, you can perform the following tasks:

- Build powerful queries without knowing SQL
- Analyze query results online, including OLAP analysis
- Edit query results to update Db2 data
- Format traditional text-based reports and reports with rich formatting
- Display charts and other complex visuals
- Send query results to an application of your choice
- Develop applications using robust API commands

How SQL statements are entered and processed

You can create your SQL statements using QMF for Workstation in several ways:

- Use the Database Explorer window to easily find and run saved queries that everyone at the same database server can share.
- If you know SQL, type the SQL statement directly in the window.
- If you don't know SQL, use the prompted or diagram interface to build the SQL statement.

The Database Explorer presents the objects that are saved on a server in a tree structure. By expanding and collapsing branches, you can easily locate and use saved queries. You can open the selected query and see the SQL statements or run the query.

If you need to build a new query, you can enter the SQL statements directly in the query window, or you can create the SQL statements using diagrams or prompts. As you build a query by using diagrams or prompts, you can open a view to see the SQL that is being created.

How you can work with query results

When you finish building the query, you can click the **Run Query** button to execute the SQL statements. After you run the query, QMF for Workstation returns the query results in an interactive window.

The query results are formatted by the comprehensive formatting options of QMF for Workstation. A robust expression language lets you conditionally format query results by retrieved column values. You can add calculated columns to the query results and group data columns on both axes with or without summaries. You can also use extensive drag-and-drop capabilities to easily restructure the appearance of the query results.

In addition to formatting the query results, you can perform the following actions:

- Create traditional text-based reports or state-of-the-art reports with rich formatting.
- Display query results by using charts and other complex visuals.
- Share reports by storing them on the database server.
- Send query results to various applications such as Microsoft Excel or Lotus® 1-2-3.

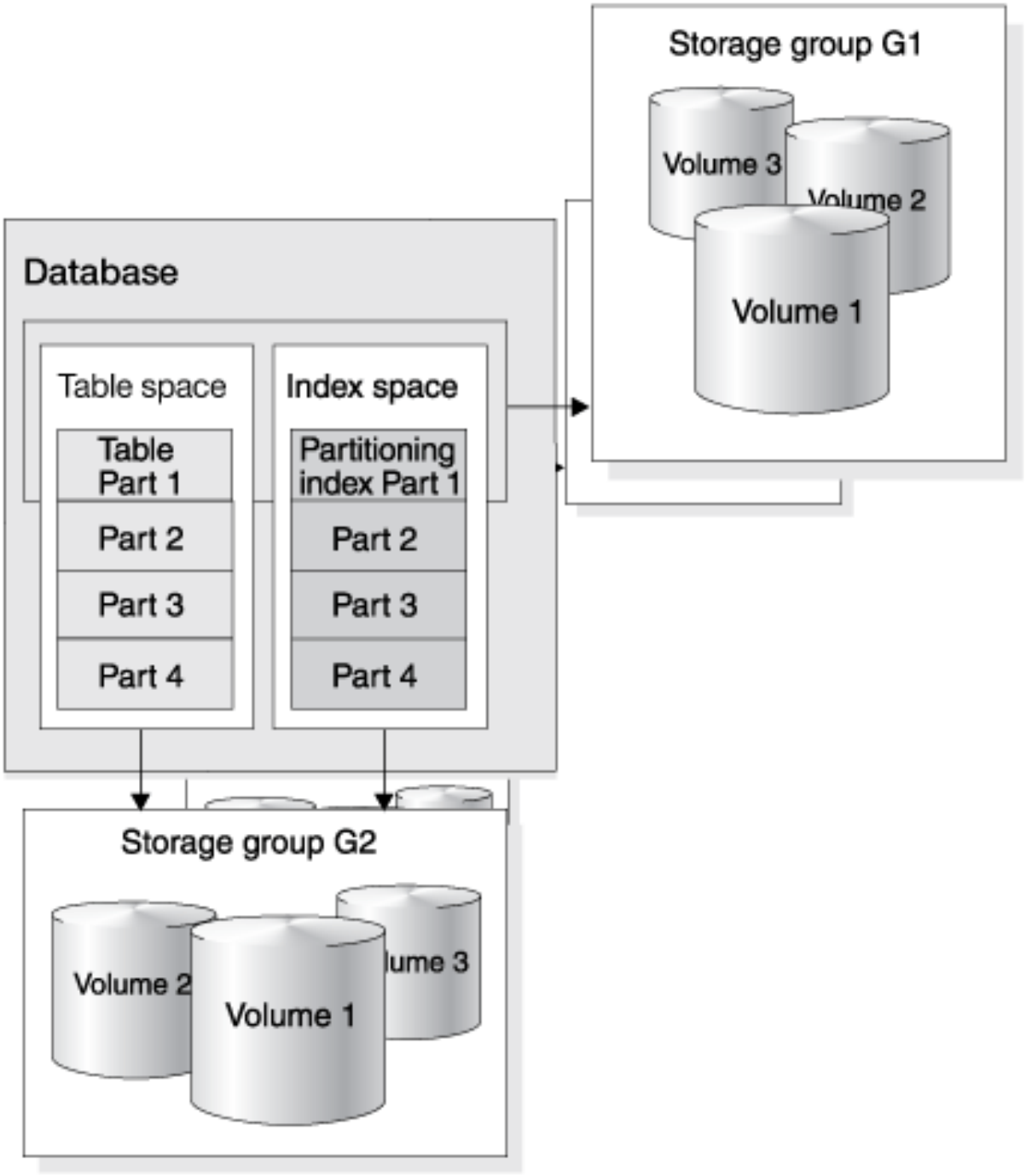
Related reference

Db2 Query Management Facility (QMF) information

Db2 database objects overview

In Db2 for z/OS, you use database objects, such as tables, table spaces, indexes, index spaces, keys, views, and databases to organize and access your data.

The brief descriptions here show how the structures fit into an overall view of Db2. The following figure shows how some Db2 structures contain others. To some extent, the notion of "containment" provides a hierarchy of structures.


Storage group G1
Volume 3
Volume 2
Volume 1
Database
Table space
Table Part 1
Part 2
Part 3
Part 4
Index space
Partitioning index Part 1
Part 2
Part 3
Part 4
Storage group G2
Volume 2
Volume 1

The Db2 structures from the most to the least inclusive are:

Databases
A set of Db2 structures that include a collection of tables, their associated indexes, and the table spaces in which they reside.

Storage groups
A set of volumes on disks that hold the data sets in which tables and indexes are stored.

Table spaces
A logical unit of storage in a database. A table space is a page set. The recommended partition-by-growth and partition-by-range table space types always contain data for only a single table. However, the segmented (non-UTS) and simple table space types, which are deprecated, can each contain one or more tables. For more information, see Table space types and characteristics in Db2 for z/OS (Db2 Administration Guide).

Deprecated function: Non-UTS table spaces for base tables are deprecated and likely to be unsupported in the future.

Tables
All data in a Db2 database is presented in *tables*, which are collections of rows all having the same columns. A table that holds persistent user data is a *base table*. A table that stores data temporarily is a *temporary table*.

Views
A *view* is an alternative way of representing data that exists in one or more tables. A view can include all or some of the columns from one or more base tables.

Indexes
An *index* is an ordered set of pointers to the data in a Db2 table. The index is stored separately from the table. An index is either a simple index or an extended index. An extended index is one of the following objects:

- An expression-based index
- A spatial index
- An XML index

Related concepts

Db2 system objects
Unlike the Db2 data structures that users create and access, Db2 controls and accesses system objects.

Implementing your database design (Db2 Administration Guide)

Storage structures
In Db2, a *storage structure* is a set of one or more VSAM data sets that hold Db2 tables or indexes. A storage structure is also called a *page set*.

Db2 schemas and schema qualifiers

The objects in a relational database are organized into sets called schemas. A *schema* is a collection of named objects that provides a logical classification of objects in the database. The first part of a schema name is the qualifier.

A schema provides a logical classification of objects in the database. The objects that a schema can contain include tables, indexes, table spaces, distinct types, functions, stored procedures, and triggers. An object is assigned to a schema when it is created.

The *schema name* of the object determines the schema to which the object belongs. A user object, such as a distinct type, function, procedure, sequence, or trigger should not be created in a *system schema*, which is any one of a set of schemas that are reserved for use by the Db2 subsystem.

When a table, index, table space, distinct type, function, stored procedure, or trigger is created, it is given a qualified two-part name. The first part is the schema name (or the qualifier), which is either implicitly or explicitly specified. The default schema is the authorization ID of the owner of the plan or package. The second part is the name of the object.

In previous versions, CREATE statements had certain restrictions when the value of CURRENT SCHEMA was different from CURRENT SQLID value. Although those restrictions no longer exist, you now must consider how to determine the qualifier and owner when CURRENT SCHEMA and CURRENT SQLID contain different values. The rules for how the owner is determined depend on the type of object being created.

CURRENT SCHEMA and CURRENT SQLID affect only dynamic SQL statements. Static CREATE statements are not affected by either CURRENT SCHEMA or CURRENT SQLID.

The following table summarizes the effect of CURRENT SCHEMA in determining the schema qualifier and owner for these objects:

- Alias
- Auxiliary table
- Created global temporary table
- Table
- View

Table 1. Schema qualifier and owner for objects

Specification of name for new object being created	Schema qualifier of new object	Owner of new object
name (no qualifier)	value of CURRENT SCHEMA	value of CURRENT SQLID
abc.name (single qualifier)	abc	abc
.....abc.name (multiple qualifiers)	abc	abc

The following table summarizes the effect of CURRENT SCHEMA in determining the schema qualifier and owner for these objects:

- User-defined distinct type
- User-defined function
- Procedure
- Sequence
- Trigger

Table 2. Schema qualifier and owner for additional objects

Specification of name for new object being created	Schema qualifier of new object	Owner of new object
name (no qualifier)	value of CURRENT SCHEMA	value of CURRENT SQLID
abc.name (single qualifier)	abc	value of CURRENT SQLID
.....abc.name (multiple qualifiers)	abc	value of CURRENT SQLID

Reserved schema names (Db2 SQL)

Db2 tables

Tables are logical structures that Db2 maintains. Db2 supports several different types of tables.

Tables are made up of columns and rows. The rows of a relational table have no fixed order. The order of the columns, however, is always the order in which you specified them when you defined the table.

At the intersection of every column and row is a specific data item, which is called a *value*. A *column* is a set of values of the same type. A *row* is a sequence of values such that the *n*th value is a value of the *n*th column of the table. Every table must have one or more columns, but the number of rows can be zero.

Db2 accesses data by referring to its content instead of to its location or organization in storage.

Db2 supports several different types of tables:

- Accelerator-only tables
- Archive tables
- Archive-enabled tables
- Auxiliary tables
- Base tables
- Clone tables
- Empty tables
- History tables
- Materialized query tables
- Result tables
- Temporal tables
- Temporary tables
- XML tables

Creation of tables
Designing tables that many applications use is a critical task. Table design can be difficult because you can represent the same information in many different ways. This information briefly describes how tables are created and altered, and how authorization is controlled.

Types of tables
In Db2, you store user data in tables. Db2 supports several types of tables, each of which has its own purpose and characteristics.

Db2 keys

A *key* is a column or an ordered collection of columns that is identified in the description of a table, an index, or a referential constraint. Keys are crucial to the table structure in a relational database.

Keys are important in a relational database because they ensure that each record in a table is uniquely identified, they help establish and enforce referential integrity, and they establish relationships between tables. The same column can be part of more than one key.

A *composite key* is an ordered set of two or more columns of the same table. The ordering of the columns is not constrained by their actual order within the table. The term *value*, when used with respect to a composite key, denotes a composite value. For example, consider this rule: "The value of the foreign key must be equal to the value of the primary key." This rule means that each component of the value of the foreign key must be equal to the corresponding component of the value of the primary key.

Db2 supports several types of keys.

Unique keys

A *unique constraint* is a rule that the values of a key are valid only if they are unique. A key that is constrained to have unique values is a *unique key*. Db2 uses a *unique index* to enforce the constraint during the execution of the LOAD utility and whenever you use an INSERT, UPDATE, or MERGE statement to add or modify data. Every unique key is a key of a unique index. You can define a unique key by using the UNIQUE clause of either the CREATE TABLE or the ALTER TABLE statement. A table can have any number of unique keys.

The columns of a unique key cannot contain null values.

Primary keys

A *primary key* is a special type of unique key and cannot contain null values. For example, the DEPTNO column in the DEPT table is a primary key.

A table can have no more than one primary key. Primary keys are optional and can be defined in CREATE TABLE or ALTER TABLE statements.

The unique index on a primary key is called a *primary index*. When a primary key is defined in a CREATE TABLE statement or ALTER TABLE statement, Db2 automatically creates the primary index.

If a unique index already exists on the columns of the primary key when it is defined in the ALTER TABLE statement, this unique index is designated as the primary index when Db2 is operating in new-function mode and implicitly created the table space.

Parent keys

A *parent key* is either a primary key or a unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the constraint.

Foreign keys

A *foreign key* is a key that is specified in the definition of a referential constraint in a CREATE or ALTER TABLE statement. A foreign key refers to or is related to a specific parent key.

Unlike other types of keys, a foreign key does not require an index on its underlying column or columns. A table can have zero or more foreign keys. The value of a composite foreign key is null if any component of the value is null.

The following figure shows the relationship between some columns in the DEPT table and the EMP table.

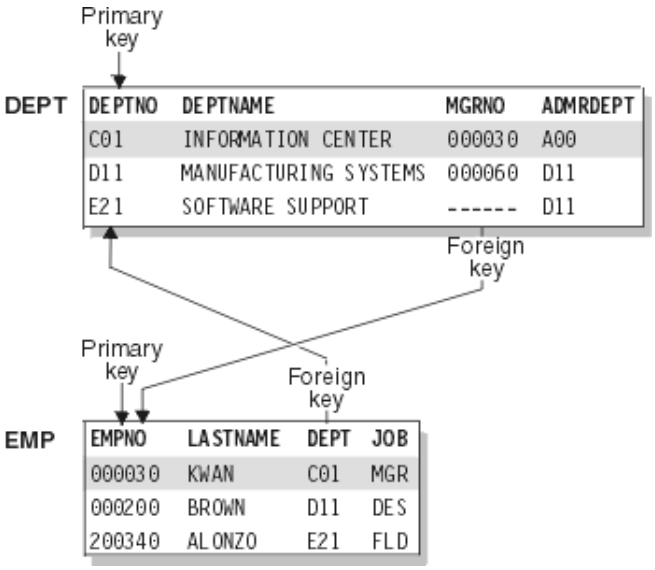


Figure 2. Relationship between DEPT and EMP tables

Figure notes: Each table has a primary key:

- DEPTNO in the DEPT table
- EMPNO in the EMP table

Each table has a foreign key that establishes a relationship between the tables:

- The values of the foreign key on the DEPT column of the EMP table match values in the DEPTNO column of the DEPT table.
- The values of the foreign key on the MGRNO column of the DEPT table match values in the EMPNO column of the EMP table when an employee is a manager.

To see a specific relationship between rows, notice how the shaded rows for department C01 and employee number 000030 share common values.

Related concepts

Referential constraints

Db2 ensures referential integrity between your tables when you define referential constraints.

Constraints

Constraints are rules that control values in columns to prevent duplicate values or set restrictions on data added to a table.

Constraints fall into the following three types:

- Unique constraints
- Referential constraints
- Check constraints

Unique constraints

A *unique constraint* is a rule that the values of a key are valid only if they are unique in a table.

Unique constraints are optional and can be defined in the CREATE TABLE or ALTER TABLE statements with the PRIMARY KEY clause or the UNIQUE clause. The columns specified in a unique constraint must be defined as NOT NULL. A unique index enforces the uniqueness of the key during changes to the columns of the unique constraint.

A table can have an arbitrary number of unique constraints, with at most one unique constraint defined as a primary key. A table cannot have more than one unique constraint on the same set of columns.

A unique constraint that is referenced by the foreign key of a referential constraint is called the *parent key*.

Referential constraints

Db2 ensures referential integrity between your tables when you define referential constraints.

Referential integrity is the state in which all values of all foreign keys are valid. Referential integrity is based on *entity integrity*. Entity integrity requires that each entity have a unique key. For example, if every row in a table represents relationships for a unique entity, the table should have one column or a set of columns that provides a unique identifier for the rows of the table. This column (or set of columns) is called the parent key of the table. To ensure that the parent key does not contain duplicate values, a unique index must be defined on the column or columns that constitute the parent key. Defining the parent key is called entity integrity.

A *referential constraint* is the rule that the nonnull values of a foreign key are valid only if they also appear as values of a parent key. The table that contains the parent key is called the *parent table* of the referential constraint, and the table that contains the foreign key is a *dependent* of that table.

The relationship between some rows of the DEPT and EMP tables, shown in the following figure, illustrates referential integrity concepts and terminology. For example, referential integrity ensures that every foreign key value in the DEPT column of the EMP table matches a primary key value in the DEPTNO column of the DEPT table.

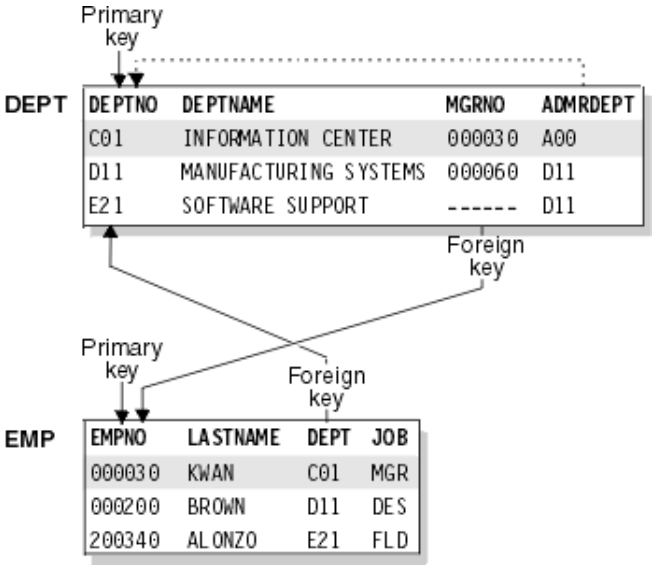


Figure 3. Referential integrity of DEPT and EMP tables

Two parent and dependent relationships exist between the DEPT and EMP tables.

- The foreign key on the DEPT column establishes a parent and dependent relationship. The DEPT column in the EMP table depends on the DEPTNO in the DEPT table. Through this foreign key relationship, the DEPT table is the parent of the EMP table. You can assign an employee to no department (by specifying a null value), but you cannot assign an employee to a department that does not exist.
- The foreign key on the MGRNO column also establishes a parent and dependent relationship. Because MGRNO depends on EMPNO, EMP is the parent table of the relationship, and DEPT is the dependent table.

You can define a primary key on one or more columns. A primary key that includes two or more columns is called a *composite key*. A foreign key can also include one or more columns. When a foreign key contains multiple columns, the corresponding primary key must be a composite key. The number of foreign key columns must be the same as the number of columns in the parent key, and the data types of the corresponding columns must be compatible. (The sample project activity table, DSN8B10.PROJACT, is an example of a table with a primary key on multiple columns, PROJNO, ACTNO, and ACSTDATE.)

A table can be a dependent of itself; this is called a *self-referencing table*. For example, the DEPT table is self-referencing because the value of the administrative department (ADMRDEPT) must be a department ID (DEPTNO). To enforce the self-referencing constraint, Db2 requires that a foreign key be defined.

Similar terminology applies to the rows of a parent-and-child relationship. A row in a dependent table, called a *dependent row*, refers to a row in a parent table, called a *parent row*. But a row of a parent table is not always a parent row—perhaps nothing refers to it. Likewise, a row of a dependent table is not always a dependent row—the foreign key can allow null values, which refer to no other rows.

Referential constraints are optional. You define referential constraints by using CREATE TABLE and ALTER TABLE statements.

Db2 enforces referential constraints when the following actions occur:

- An INSERT statement is applied to a dependent table.
- An UPDATE statement is applied to a foreign key of a dependent table or to the parent key of a parent table.
- A MERGE statement that includes an insert operation is applied to a dependent table.
- A MERGE statement that includes an update operation is applied to a foreign key of a dependent table or to the parent key of a parent table.
- A DELETE statement is applied to a parent table. All affected referential constraints and all delete rules of all affected relationships must be satisfied in order for the delete operation to succeed.

- The LOAD utility with the ENFORCE CONSTRAINTS option is run on a dependent table.
- The CHECK DATA utility is run.

Another type of referential constraint is an *informational referential constraint*. This type of constraint is not enforced by Db2 during normal operations. An application process should verify the data in the referential integrity relationship. An informational referential constraint allows queries to take advantage of materialized query tables.

The order in which referential constraints are enforced is undefined. To ensure that the order does not affect the result of the operation, there are restrictions on the definition of delete rules and on the use of certain statements. The restrictions are specified in the descriptions of the SQL statements CREATE TABLE, ALTER TABLE, INSERT, UPDATE, MERGE, and DELETE.

The rules of referential integrity involve the following concepts and terminology:

parent key
A primary key or a unique key of a referential constraint.

parent table
A table that is a parent in at least one referential constraint. A table can be defined as a parent in an arbitrary number of referential constraints.

dependent table
A table that is a dependent in at least one referential constraint. A table can be defined as a dependent in an arbitrary number of referential constraints. A dependent table can also be a parent table.

descendent table
A table that is a dependent of another table or a table that is a dependent of a descendent table.

referential cycle
A set of referential constraints in which each associated table is a descendent of itself.

parent row
A row that has at least one dependent row.

dependent row
A row that has at least one parent row.

descendent row
A row that is dependent on another row or a row that is a dependent of a descendent row.

self-referencing row
A row that is a parent of itself.

self-referencing table
A table that is both parent and dependent in the same referential constraint. The constraint is called a *self-referencing constraint*.

A temporal referential constraint can be defined for a table that contains a BUSINESS_TIME period. The PERIOD BUSINESS_TIME clause is used in both the FOREIGN KEY clause and the REFERENCES clause to indicate that there must not be a row in the child table for which the period of time represented by the BUSINESS_TIME period value for that row is not contained in the BUSINESS_TIME period of one or more corresponding rows in the parent table. Unlike normal referential constraints, it is not necessary that there be exactly one corresponding row in the parent table where the BUSINESS_TIME period contains the BUSINESS_TIME period of the child row. As long as the BUSINESS_TIME period of a row in the child table is contained in the union of the BUSINESS_TIME periods of two or more contiguous matching rows in the parent table, the temporal referential constraint is satisfied.

Additionally, the following indexes must be defined:

- A unique index on the parent table with the BUSINESS_TIME WITHOUT OVERLAPS clause.
- A non-unique index on the child table with the BUSINESS_TIME WITH OVERLAPS clause. Alternatively, the index on the child table, can be defined without the BUSINESS_TIME WITH OVERLAPS clause if the end of the index key includes the end column followed by the begin column of the BUSINESS_TIME period (both in ascending order).

The following rules provide referential integrity:

insert rule
A nonnull insert value of the foreign key must match some value of the parent key of the parent table. The value of a composite foreign key is null if any component of the value is null.

update rule
A nonnull update value of the foreign key must match some value of the parent key of the parent table. The value of a composite foreign key is treated as null if any component of the value is null.

delete rule
Controls what happens when a row of the parent table is deleted. The choices of action, made when the referential constraint is defined, are RESTRICT, NO ACTION, CASCADE, or SET NULL. SET NULL can be specified only if some column of the foreign key allows null values.

More precisely, the delete rule applies when a row of the parent table is the object of a delete or propagated delete operation and that row has dependents in the dependent table of the referential constraint. A *propagated delete* refers to the situation where dependent rows are deleted when parent rows are deleted. Let *P* denote the parent table, let *D* denote the dependent table, and let *p* denote a parent row that is the object of a delete or propagated delete operation. If the delete rule is:

- RESTRICT or NO ACTION, an error occurs and no rows are deleted.
- CASCADE, the delete operation is propagated to the dependent rows of *p* in *D*.
- SET NULL, each nullable column of the foreign key of each dependent row of *p* in *D* is set to null.

Each referential constraint in which a table is a parent has its own delete rule, and all applicable delete rules are used to determine the result of a delete operation. Thus, a row cannot be deleted if it has dependents in a referential constraint with a delete rule of RESTRICT or NO ACTION or the deletion cascades to any of its descendents that are dependents in a referential constraint with the delete rule of RESTRICT or NO ACTION.

The deletion of a row from parent table *P* involves other tables and can affect rows of these tables:

- If *D* is a dependent of *P* and the delete rule is RESTRICT or NO ACTION, *D* is involved in the operation but is not affected by the operation and the deletion from the parent table *P* does not take place.
- If *D* is a dependent of *P* and the delete rule is SET NULL, *D* is involved in the operation and rows of *D* might be updated during the operation.
- If *D* is a dependent of *P* and the delete rule is CASCADE, *D* is involved in the operation and rows of *D* might be deleted during the operation. If rows of *D* are deleted, the delete operation on *P* is said to be propagated to *D*. If *D* is also a parent table, the actions described in this list apply, in turn, to the dependents of *D*.

Any table that can be involved in a delete operation on *P* is said to be *delete-connected* to *P*. Thus, a table is delete-connected to table *P* if it is a dependent of *P* or a dependent of a table to which delete operations from *P* cascade.

Department table (DSN8B10.DEPT)
The department table describes each department in the enterprise and identifies its manager and the department to which it reports.

Employee table (DSN8B10.EMP)
The sample employee table identifies all employees by an employee number and lists basic personnel information.

Project activity table (DSN8B10.PROJACT)
The sample project activity table lists the activities that are performed for each project.

Referential constraints (Db2 Application programming and SQL)

Check constraints

A *check constraint* is a rule that specifies the values that are allowed in one or more columns of every row of a base table.

Like referential constraints, check constraints are optional and you define them by using the CREATE TABLE and ALTER TABLE statements. The definition of a check constraint restricts the values that a specific column of a base table can contain.

A table can have any number of check constraints. Db2 enforces a check constraint by applying the restriction to each row that is inserted, loaded, or updated. One restriction is that a column name in a check constraint on a table must identify a column of that table.

Example: You can create a check constraint to ensure that all employees earn a salary of $30,000 or more:

```
CHECK (SALARY>= 30000)
```

Related concepts

Check constraints (Db2 Application programming and SQL)

Db2 columns

A column definition has two basic components, the column name and the data type.

A *column* contains values that have the same data type. If you are familiar with the concepts of records and fields, you can think of a *value* as a field in a record. A value is the smallest unit of data that you can manipulate with SQL. For example, in the EMP table, the EMPNO column identifies all employees by a unique employee number. The HIREDATE column contains the hire dates for all employees. You cannot overlap columns.

Online schema enhancements provide flexibility that lets you change a column definition. Carefully consider the decisions that you make about column definitions. After you implement the design of your tables, you can change a column definition with minimal disruption of applications.

Throughout the implementation phase of database design, refer to the complete descriptions of SQL statement syntax and usage for each SQL statement that you work with.

Column names

Following column naming guidelines that are developed for your organization ensures that you make good choices that are consistent.

Generally, the database administrator (DBA) is involved in determining the names of attributes (or columns) during the physical database design phase. To make the right choices for column names, DBAs follow the guidelines that the data administrators developed.

Sometimes columns need to be added to the database after the design is complete. In this case, Db2 rules for unique column names must be followed. Column names must be unique within a table, but you can use the same column name in different tables. Try to choose a meaningful name to describe the data in a column to make your naming scheme intuitive. The maximum length of a column name is 30 bytes.

Related concepts

Column names (Db2 SQL)

Data types (Db2 SQL)

Related tasks

Altering the data type of a column (Db2 Administration Guide)

Related reference

Employee table (DSN8B10.EMP)

The sample employee table identifies all employees by an employee number and lists basic personnel information.

Data types

Every column in every Db2 table has a data type. The data type influences the range of values that the column can have and the set of operators and functions that apply to it.

You specify the data type of each column at the time that you create the table. You can also change the data type of a table column. The new data type definition is applied to all data in the associated table when the table is reorganized.

Some data types have parameters that further define the operators and functions that apply to the column. Db2 supports both IBM-supplied data types and user-defined data types. The data types that IBM supplies are sometimes called *built-in data types*.

In Db2 for z/OS, user-defined data types are called *distinct types*.

How Db2 compares values of different data types

Db2 compares values of different types and lengths. A comparison occurs when both values are numeric, both values are character strings, or both values are graphic strings. Comparisons can also occur between character and graphic data or between character and datetime data if the character data is a valid character representation of a datetime value. Different types of string or numeric comparisons might have an impact on performance.

Related concepts

Entity attributes in database design

When you define attributes for entities, you generally work with the data administrator to decide on names, data types, and appropriate values for the attributes.

Data types of columns (Db2 Application programming and SQL)

Assignment and comparison (Db2 SQL)

Casting between data types (Db2 SQL)

Rules for result data types (Db2 SQL)

Distinct types

A *distinct type* is a user-defined data type that is based on existing built-in Db2 data types.

String data types

Db2 supports several types of string data: character strings, graphic strings, and binary strings.

Character strings contain text and can be either a fixed-length or a varying-length. *Graphic strings* contain graphic data, which can also be either a fixed-length or a varying-length. *Binary strings* contain strings of binary bytes and can be either a fixed-length or a varying-length. All of these types of string data can be represented as large objects.

The following table describes the different string data types and indicates the range for the length of each string data type.

Table 3. String data types

Data type	Denotes a column of...
CHARACTER(*n*)	Fixed-length character strings with a length of *n* bytes. *n* must be greater than 0 and not greater than 255. The default length is 1.
VARCHAR(*n*)	Varying-length character strings with a maximum length of *n* bytes. *n* must be greater than 0 and less than a number that depends on the page size of the table space. The maximum length is 32704.
CLOB(*n*)	Varying-length character strings with a maximum of *n* characters. *n* cannot exceed 2,147,483,647. The default length is 1M.

Table 3. String data types (continued)

Data type	Denotes a column of...
GRAPHIC(*n*)	Fixed-length graphic strings that contain *n* double-byte characters. *n* must be greater than 0 and less than 128. The default length is 1.
VARGRAPHIC(*n*)	Varying-length graphic strings. The maximum length, *n*, must be greater than 0 and less than a number that depends on the page size of the table space. The maximum length is 16352.
DBCLOB(*n*)	Varying-length strings of double-byte characters with a maximum of *n* double-byte characters. *n* cannot exceed 1,073,741,824. The default length is 1M.
BINARY(*n*)	Fixed-length or varying-length binary strings with a length of *n* bytes. *n* must be greater than 0 and not greater than 255. The default length is 1.
VARBINARY(*n*)	Varying-length binary strings with a length of *n* bytes. The length of *n* must be greater than 0 and less than a number that depends on the page size of the table space. The maximum length is 32704.
BLOB(*n*)	Varying-length binary strings with a length of *n* bytes. *n* cannot exceed 2,147,483,647. The default length is 1M.

In most cases, the content of the data that a column is to store dictates the data type that you choose.

Example: The DEPT table has a column, DEPTNAME. The data type of the DEPTNAME column is VARCHAR(36). Because department names normally vary considerably in length, the choice of a varying-length data type seems appropriate. If you choose a data type of CHAR(36), for example, the result is a lot of wasted, unused space. In this case, Db2 assigns all department names, regardless of length, the same amount of space (36 bytes). A data type of CHAR(6) for the employee number (EMPNO) is a reasonable choice because all values are fixed-length values (6 bytes).

Fixed-length and variable-length character strings

Using VARCHAR saves disk space, but it incurs a 2-byte overhead cost for each value. Using VARCHAR also requires additional processing for varying-length rows. Therefore, using CHAR is preferable to VARCHAR, unless the space that you save by using VARCHAR is significant. The savings are not significant if the maximum column length is small or if the lengths of the values do not have a significant variation.

Recommendation: Generally, do not define a column as VARCHAR(*n*) or CLOB(*n*) unless *n* is at least 18 characters.

String subtypes

If an application that accesses your table uses a different encoding scheme than your DBMS uses, the following string subtypes can be important:

BIT
Does not represent characters.

SBCS
Represents single-byte characters.

MIXED
Represents single-byte characters and multibyte characters.

String subtypes apply only to CHAR, VARCHAR, and CLOB data types. However, the BIT string subtype is not allowed for the CLOB data type.

Graphic and mixed data

When columns contain *double-byte character set* (DBCS) characters, you can define them as either graphic data or mixed data.

Graphic data can be either GRAPHIC, VARGRAPHIC, or DBCLOB. Using VARGRAPHIC saves disk space, but it incurs a 2-byte overhead cost for each value. Using VARGRAPHIC also requires additional processing for varying-length rows. Therefore, using GRAPHIC data is preferable to using VARGRAPHIC unless the space that you save by using VARGRAPHIC is significant. The savings are not significant if the maximum column length is small or if the lengths of the values do not vary significantly.

Recommendation: Generally, do not define a column as VARGRAPHIC(*n*) unless *n* is at least 18 double-byte characters (which is a length of 36 bytes).

Mixed-data character string columns can contain both *single-byte character set* (SBCS) and DBCS characters. You can specify the mixed-data character string columns as CHAR, VARCHAR, or CLOB with MIXED DATA.

Recommendation: If all of the characters are DBCS characters, use the graphic data types. (Kanji is an example of a language that requires DBCS characters.) For SBCS characters, use mixed data to save 1 byte for every single-byte character in the column.

Encoding schemes for string data

For string data, all characters are represented by a common encoding representation (Unicode, ASCII, or EBCDIC). Encoding schemes apply to string data types and to distinct types that are based on string types.

Multinational companies that engage in international trade often store data from more than one country in the same table. Some countries use different coded character set identifiers. Db2 for z/OS supports the Unicode encoding scheme, which represents many different geographies and languages. If you need to perform character conversion on Unicode data, the conversion is more likely to preserve all of your information.

In some cases, you might need to convert characters to a different encoding representation. The process of conversion is known as *character conversion*. Most users do not need a knowledge of character conversion. When character conversion does occur, it does so automatically and a successful conversion is invisible to the application and users.

Related concepts

Distinct types
A *distinct type* is a user-defined data type that is based on existing built-in Db2 data types.

Character strings (Db2 SQL)

Graphic strings (Db2 SQL)

Binary strings (Db2 SQL)

Introduction to character conversion (Db2 Internationalization Guide)

Related reference

Department table (DSN8B10.DEPT)
The department table describes each department in the enterprise and identifies its manager and the department to which it reports.

Numeric data types

Db2 supports several types of numeric data types, each of which has its own characteristics.

For numeric data, use numeric columns rather than string columns. Numeric columns require less space than string columns, and Db2 verifies that the data has the assigned type.

Example: Assume that Db2 calculates a range between two numbers. If the values have a string data type, Db2 assumes that the values can include all combinations of alphanumeric characters. In contrast, if the values have a numeric data type, Db2 can calculate a range between the two values more efficiently.

The following table describes the numeric data types.

Table 4. Numeric data types

Data type	Denotes a column of...
SMALLINT	Small integers. A *small integer* is binary integer with a precision of 15 bits. The range is -32768 to +32767.
INTEGER or INT	Large integers. A *large integer* is binary integer with a precision of 31 bits. The range is -2147483648 to +2147483647.
BIGINT	Big integers. A *big integer* is a binary integer with a precision of 63 bits. The range of big integers is -9223372036854775808 to +9223372036854775807.
DECIMAL or NUMERIC	A *decimal* number is a packed decimal number with an implicit decimal point. The position of the decimal point is determined by the precision and the scale of the number. The scale, which is the number of digits in the fractional part of the number, cannot be negative or greater than the precision. The maximum precision is 31 digits. All values of a decimal column have the same precision and scale. The range of a decimal variable or the numbers in a decimal column is -*n* to +*n*, where *n* is the largest positive number that can be represented with the applicable precision and scale. The maximum range is 1 - 10^{31} to 10^{31} - 1.
DECFLOAT	A *decimal floating-point* value is an IEEE 754r number with a decimal point. The position of the decimal point is stored in each decimal floating-point value. The maximum precision is 34 digits. The range of a decimal floating-point number is either 16 or 34 digits of precision; the exponent range is respectively 10-383 to 10+384 or 10-6143 to 10+6144.
REAL	A *single-precision floating-point* number is a short floating-point number of 32 bits. The range of single-precision floating-point numbers is approximately -7.2E+75 to 7.2E+75. In this range, the largest negative value is about -5.4E-79, and the smallest positive value is about 5.4E-079.
DOUBLE	A *double-precision floating-point* number is a long floating-point number of 64-bits. The range of double-precision floating-point numbers is approximately -7.2E+75 to 7.2E+75. In this range, the largest negative value is about -5.4E-79, and the smallest positive value is about 5.4E-79.

Note: IBM Z and z/Architecture® use the System/390® format and support IEEE floating-point format.

For integer values, use SMALLINT, INTEGER, or BIGINT (depending on the range of the values). Do not use DECIMAL for integer values.

You can define an exact numeric column as an identity column. An *identity column* has an attribute that enables Db2 to automatically generate a unique numeric value for each row that is inserted into the table. Identity columns are ideally suited to the task of generating unique primary-key values. Applications that use identity columns might be able to avoid concurrency and performance problems that sometimes occur when applications implement their own unique counters.

Date, time, and timestamp data types

Although storing dates and times as numeric values is possible, using datetime data types is recommended. The datetime data types are DATE, TIME, and TIMESTAMP.

The following table describes the data types for dates, times, and timestamps.

Table 5. Date, time, and timestamp data types

Data type	Denotes a column of...
DATE	A *date* is a three-part value representing a year, month, and day in the range of 0001-01-01 to 9999-12-31.
TIME	A *time* is a three-part value representing a time of day in hours, minutes, and seconds, in the range of 00.00.00 to 24.00.00.
TIMESTAMP	A *timestamp* is a seven-part value representing a date and time by year, month, day, hour, minute, second, and microsecond, in the range of 0001-01-01-00.00.00.000000000 to 9999-12-31-24.00.00.000000000 with nanosecond precision. Timestamps can also hold timezone information.

Db2 stores values of datetime data types in a special internal format. When you load or retrieve data, Db2 can convert it to or from any of the formats in the following table.

Table 6. Date and time format options

Format name	Abbreviation	Typical date	Typical time
International Standards Organization	ISO	2003-12-25	13.30.05
IBM USA standard	USA	12/25/2003	1:30 PM
IBM European standard	EUR	25.12.2003	13.30.05
Japanese Industrial Standard Christian Era	JIS	2003-12-25	13:30:05

GUPI

Example 1: The following query displays the dates on which all employees were hired, in IBM USA standard form, regardless of the local default:

```
SELECT EMPNO, CHAR(HIREDATE, USA) FROM EMP;
```

When you use datetime data types, you can take advantage of Db2 built-in functions that operate specifically on datetime values, and you can specify calculations for datetime values.

Example 2: Assume that a manufacturing company has an objective to ship all customer orders within five days. You define the SHIPDATE and ORDERDATE columns as DATE data types. The company can use datetime data types and the DAYS built-in function to compare the shipment date to the order date. Here is how the company might code the function to generate a list of orders that have exceeded the five-day shipment objective:

```
DAYS(SHIPDATE) – DAYS(ORDERDATE)> 5
```

As a result, programmers do not need to develop, test, and maintain application code to perform complex datetime arithmetic that needs to allow for the number of days in each month.

GUPI

You can use the following sample user-defined functions (which come with Db2) to modify the way dates and times are displayed.

- ALTDATE returns the current date in a user-specified format or converts a user-specified date from one format to another.
- ALTTIME returns the current time in a user-specified format or converts a user-specified time from one format to another.

At installation time, you can also supply an exit routine to make conversions to and from any local standard.

When loading date or time values from an outside source, Db2 accepts any of the date and time format options that are listed in this information. Db2 converts valid input values to the internal format. For retrieval, a default format is specified at Db2 installation time. You can subsequently override that default by using a precompiler option for all statements in a program or by using the scalar function CHAR for a particular SQL statement and by specifying the format that you want.

Related concepts

Datetime constants (Db2 SQL)

XML data type

The XML data type is used to define columns of a table that store XML values. This pureXML® data type provides the ability to store well-formed XML documents in a database.

All XML data is stored in the database in an internal representation. Character data in this internal representation is in the UTF-8 encoding scheme.

XML values that are stored in an XML column have an internal representation that is not a string and not directly comparable to string values. An XML value can be transformed into a serialized string value that represents the XML document by using the XMLSERIALIZE function or by retrieving the value into an application variable of an XML, string, or binary type. Similarly, a string value that represents an XML document can be transformed to an XML value by using the XMLPARSE function or by storing a value from a string, binary, or XML application data type in an XML column.

The size of an XML value in a Db2 table has no architectural limit. However, serialized XML data that is stored in or retrieved from an XML column is limited to 2 GB.

Validation of an XML document against an XML schema, typically performed during INSERT or UPDATE into an XML column, is supported by the XML schema repository (XSR). If an XML column has an XML type modifier, documents that are inserted into the column or updated in the column are automatically validated against an XML schema.

Large object data types

You can use large object data types to store audio, video, images, and other files that are larger than 32 KB.

The VARCHAR, VARGRAPHIC, and VARBINARY data types have a storage limit of 32 KB. However, applications often need to store large text documents or additional data types such as audio, video, drawings, images, and a combination of text and graphics. For data objects that are larger than 32 KB, you can use the corresponding large object (LOB) data types to store these objects.

Db2 provides three data types to store these data objects as strings of up to 2 GB in size:

Character large objects (CLOBs)
: Use the CLOB data type to store SBCS or mixed data, such as documents that contain single character set. Use this data type if your data is larger (or might grow larger) than the VARCHAR data type permits.

Double-byte character large objects (DBCLOBs)
: Use the DBCLOB data type to store large amounts of DBCS data, such as documents that use a DBCS character set.

Binary large objects (BLOBs)
: Use the BLOB data type to store large amounts of noncharacter data, such as pictures, voice, and mixed media.

If your data does not fit entirely within a data page, you can define one or more columns as LOB columns. An advantage to using LOBs is that you can create user-defined functions that are allowed only on LOB data types.

Related concepts

Creation of large objects
Defining large objects to Db2 is different than defining other types of data and objects.

Large objects (LOBs) (Db2 SQL)

ROWID data type

You use the ROWID data type to uniquely identify rows in a Db2 subsystem.

Db2 can generate a value for the column when a row is added, depending on the option that you choose (GENERATED ALWAYS or GENERATED BY DEFAULT) when you define the column. You can use a ROWID column in a table for several reasons.

- You can define a ROWID column to include LOB data in a table.
- You can use direct-row access so that Db2 accesses a row directly through the ROWID column. If an application selects a row from a table that contains a ROWID column, the row ID value implicitly contains the location of the row. If you use that row ID value in the search condition of subsequent SELECT statements, Db2 might be able to navigate directly to the row.

  Requirement: To use direct row access, you must use a retrieved ROWID value before you commit. When your application commits, it releases its claim on the table space. After the commit, if a REORG is run on your table space, the physical location of the rows might change.
- You can define a ROWID column with the IMPLICITLY HIDDEN attribute. Such columns are returned only for SQL statements that explicitly name the column. SQL statements that only imply selection of the column, such as statements that specify SELECT *, do not return the column.

Related concepts

Row ID values (Db2 SQL)

Direct row access (PRIMARY_ACCESSTYPE='D') (Db2 Performance)

Related tasks

Specifying direct row access by using row IDs (Db2 Application programming and SQL)

Related reference

ROWID (Db2 SQL)

Distinct types

A *distinct type* is a user-defined data type that is based on existing built-in Db2 data types.

A distinct type is internally the same as a built-in data type, but Db2 treats them as a separate and incompatible type for semantic purposes.

Defining your own distinct type ensures that only functions that are explicitly defined on a distinct type can be applied to its instances.

Example 1: You might define a US_DOLLAR distinct type that is based on the Db2 DECIMAL data type to identify decimal values that represent United States dollars. The US_DOLLAR distinct type does not automatically acquire the functions and operators of its source type, DECIMAL.

Although you can have different distinct types that are based on the same built-in data types, distinct types have the property of *strong typing*. With this property, you cannot directly compare instances of a distinct type with anything other than another instance of that same type. Strong typing prevents semantically incorrect operations (such as explicit addition of two different currencies) without first undergoing a conversion process. You define which types of operations can occur for instances of a distinct type.

If your company wants to track sales in many countries, you must convert the currency for each country in which you have sales.

GUPI

Example 2: You can define a distinct type for each country. For example, to create US_DOLLAR types and CANADIAN_DOLLAR types, you can use the following CREATE DISTINCT TYPE statements:

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9,2);
CREATE DISTINCT TYPE CANADIAN_DOLLAR AS DECIMAL (9,2);
```

Example 3: After you define distinct types, you can use them in your CREATE TABLE statements:

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM_NO INTEGER,
```

```
   MONTH           INTEGER,
   YEAR            INTEGER,
   TOTAL_AMOUNT    US_DOLLAR);
CREATE TABLE CANADIAN_SALES
  (PRODUCT_ITEM_NO INTEGER,
   MONTH           INTEGER,
   YEAR            INTEGER,
   TOTAL_AMOUNT    CANADIAN_DOLLAR);
```

GUPI

User-defined functions support the manipulation of distinct types.

Related concepts

String data types

Db2 supports several types of string data: character strings, graphic strings, and binary strings.

Distinct types (Db2 SQL)

Distinct type assignments (Db2 SQL)

Distinct type comparisons (Db2 SQL)

Null values

Some columns cannot have a meaningful value in every row. Db2 uses a special value indicator, the *null value*, to stand for an unknown or missing value. A null value is a special value that Db2 interprets to mean that no data is present.

If you do not specify otherwise,Db2 allows any column to contain null values. Users can create rows in the table without providing a value for the column.

Using the NOT NULL clause enables you to disallow null values in the column. Primary keys must be defined as NOT NULL.

GUPI

Example: The table definition for the DEPT table specifies when you can use a null value. Notice that you can use nulls for the MGRNO column only:

```
CREATE TABLE DEPT
      (DEPTNO    CHAR(3)           NOT NULL,
       DEPTNAME  VARCHAR(36)       NOT NULL,
       MGRNO     CHAR(6)                    ,
       ADMRDEPT  CHAR(3)           NOT NULL,
       PRIMARY KEY (DEPTNO)                 )
  IN MYDB.MYTS;
```

GUPI

Before you decide whether to allow nulls for unknown values in a particular column, you must be aware of how nulls affect results of a query:

- Nulls in application programs

  Nulls do not satisfy any condition in an SQL statement other than the special IS NULL predicate. Db2 sorts null values differently than non-null values. Null values do not behave like other values. For example, if you ask Db2 whether a null value is larger than a given known value, the answer is UNKNOWN. If you then ask Db2 whether a null value is smaller than the same known value, the answer is still UNKNOWN.

  If getting a value of UNKNOWN is unacceptable for a particular column, you could define a default value instead. Programmers are familiar with the way default values behave.

- Nulls in a join operation

  Nulls need special handling in join operations. If you perform a join operation on a column that can contain null values, consider using an outer join.

Related concepts

Comparison of null values and default values

Using a null value is easier and better than using a default value in some situations.

Nulls (Db2 SQL)

Ways to join data from more than one table

When you want to see information from multiple tables, you can use a SELECT statement. SELECT statements can retrieve and join column values from two or more tables into a single row. The retrieval is based on a specified condition, typically of matching column values.

Entity attributes in database design

When you define attributes for entities, you generally work with the data administrator to decide on names, data types, and appropriate values for the attributes.

Default values

Db2 defines some default values, and you define others (by using the DEFAULT clause in the CREATE TABLE or ALTER TABLE statement).

If a column is defined as NOT NULL WITH DEFAULT or if you do not specify NOT NULL, Db2 stores a default value for a column whenever an insert or load does not provide a value for that column. If a column is defined as NOT NULL, Db2 does not supply a default value.

Db2–defined default values

Db2 generates a default value for ROWID columns. Db2 also determines default values for columns that users define with NOT NULL WITH DEFAULT, but for which no specific value is specified, as shown in the following table.

Table 7. Db2-defined default values for data types

For columns of...	Data types	Default
Numbers	SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, REAL, DOUBLE, DECFLOAT, or FLOAT	0
Fixed-length strings	CHAR or GRAPHIC	Blanks
	BINARY	Hexadecimal zeros
Varying-length strings	VARCHAR, CLOB, VARGRAPHIC, DBCLOB, VARBINARY, or BLOB	Empty string
Dates	DATE	CURRENT DATE
Times	TIME	CURRENT TIME
Timestamps	TIMESTAMP	CURRENT TIMESTAMP
ROWIDs	ROWID	Db2-generated

User-defined default values

You can specify a particular default value, such as:

```
DEFAULT 'N/A'
```

When you choose a default value, you must be able to assign it to the data type of the column. For example, all string constants are VARCHAR. You can use a VARCHAR string constant as the default for a CHAR column even though the type isn't an exact match. However, you could not specify a default value of 'N⁄A' for a column with a numeric data type.

In the next example, the columns are defined as CHAR (fixed length). The special registers (USER and CURRENT SQLID) that are referenced contain varying length values.

Example: If you want a record of each user who inserts any row of a table, define the table with two additional columns:

```
PRIMARY_ID      CHAR(8)     WITH DEFAULT USER,
SQL_ID          CHAR(8)     WITH DEFAULT CURRENT SQLID,
```

You can then create a view that omits those columns and allows users to update the view instead of the base table. Db2 then adds, by default, the primary authorization ID and the SQLID of the process.

When you add columns to an existing table, you must define them as nullable or as not null with default. Assume that you add a column to an existing table and specify not null with default. If Db2 reads from the table before you add data to the column, the column values that you retrieve are the default values. With few exceptions, the default values for retrieval are the same as the default values for insert.

Default values for ROWID

Db2 always generates the default values for ROWID columns.

Related concepts

Comparison of null values and default values
Using a null value is easier and better than using a default value in some situations.

Authorization and security mechanisms for data access
Authorization is an important part of controlling Db2. The security and authorization mechanisms that control access to Db2 data are both direct and indirect.

Related reference

ALTER TABLE (Db2 SQL)
CREATE TABLE (Db2 SQL)

Comparison of null values and default values

Using a null value is easier and better than using a default value in some situations.

Suppose that you want to find out the average salary for all employees in a department. The salary column does not always need to contain a meaningful value, so you can choose between the following options:

- Allowing null values for the SALARY column
- Using a non-null default value (such as, 0)

By allowing null values, you can formulate the query easily, and Db2 provides the average of all known or recorded salaries. The calculation does not include the rows that contain null values. In the second case, you probably get a misleading answer unless you know the nonnull default value for unknown salaries and formulate your query accordingly.

The following figure shows two scenarios. The table in the figure excludes salary data for employee number 200440, because the company just hired this employee and has not yet determined the salary. The calculation of the average salary for department E21 varies, depending on whether you use null values or nonnull default values.

- The left side of the figure assumes that you use null values. In this case, the calculation of average salary for department E21 includes only the three employees (000320, 000330, and 200340) for whom salary data is available.
- The right side of the figure assumes that you use a nonnull default value of zero (0). In this case, the calculation of average salary for department E21 includes all four employees, although valid salary information is available for only three employees.

As you can see, only the use of a null value results in an accurate average salary for department E21.

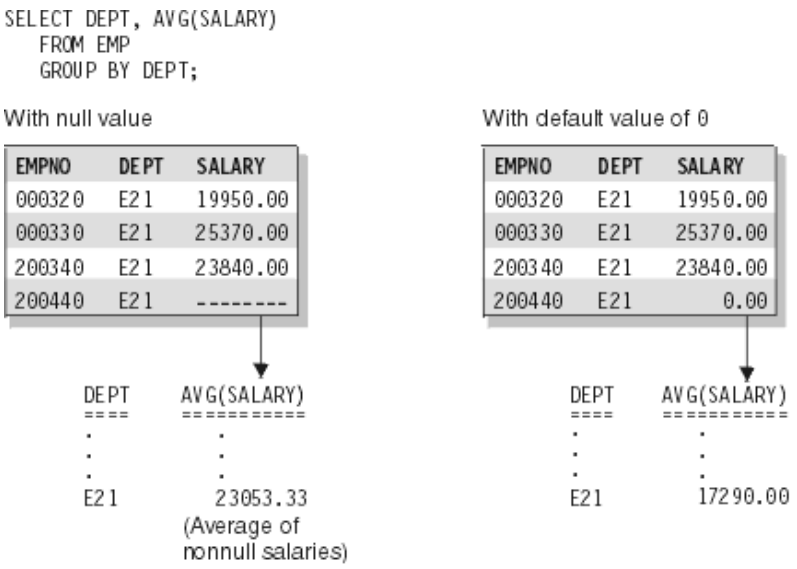


Figure 4. When nulls are preferable to default values

Null values are distinct in most situations so that two null values are not equal to each other.

GUPI

Example: The following example shows how to compare two columns to see if they are equal or if both columns are null:

```
WHERE E1.DEPT IS NOT DISTINCT FROM E2.DEPT
```

GUPI

Related concepts

Null values

Some columns cannot have a meaningful value in every row. Db2 uses a special value indicator, the *null value*, to stand for an unknown or missing value. A null value is a special value that Db2 interprets to mean that no data is present.

Default values

Db2 defines some default values, and you define others (by using the DEFAULT clause in the CREATE TABLE or ALTER TABLE statement).

Assignment and comparison (Db2 SQL)

Use of check constraints to enforce validity of column values

You can use check constraints to ensure that only values from the domain for the column or attribute are allowed.

As a result of using check constraints, programmers do not need to develop, test, and maintain application code that performs these checks.

You can choose to define check constraints by using the SQL CREATE TABLE statement or ALTER TABLE statement. For example, you might want to ensure that each value in the SALARY column of the EMP table contains more than a certain minimum amount.

Db2 enforces a check constraint by applying the relevant search condition to each row that is inserted, updated, or loaded. An error occurs if the result of the search condition is false for any row.

Use of check constraints to insert rows into tables

When you use the INSERT statement or the MERGE statement to add a row to a table, Db2 automatically enforces all check constraints for that table. If the data violates any check constraint that is defined on that table, Db2 does not insert the row.

GUPI

Example 1: Assume that the NEWEMP table has the following two check constraints:

- Employees cannot receive a commission that is greater than their salary.
- Department numbers must be between '001' to '100,' inclusive.

Consider this INSERT statement, which adds an employee who has a salary of $65,000 and a commission of $6000:

```
INSERT INTO NEWEMP
  (EMPNO, FIRSTNME, LASTNAME, DEPT, JOB, SALARY, COMM)
  VALUES ('100125', 'MARY', 'SMITH','055', 'SLS', 65000.00, 6000.00);
```

The INSERT statement in this example succeeds because it satisfies both constraints.

Example 2: Consider this INSERT statement:

```
INSERT INTO NEWEMP
  (EMPNO, FIRSTNME, LASTNAME, DEPT, JOB, SALARY, COMM)
  VALUES ('120026', 'JOHN', 'SMITH','055', 'DES', 5000.00, 55000.00 );
```

The INSERT statement in this example fails because the $55,000 commission is higher than the $5,000 salary. This INSERT statement violates a check constraint on NEWEMP.

GUPI

Use of check constraints to update tables

Db2 automatically enforces all check constraints for a table when you use the UPDATE statement or the MERGE statement to change a row in the table. If the intended update violates any check constraint that is defined on that table, Db2 does not update the row.

GUPI

Example: Assume that the NEWEMP table has the following two check constraints:

- Employees cannot receive a commission that is greater than their salary.
- Department numbers must be between '001' to '100,' inclusive.

Consider this UPDATE statement:

```
UPDATE NEWEMP
  SET DEPT = '011'
  WHERE FIRSTNME = 'MARY' AND LASTNAME= 'SMITH';
```

This update succeeds because it satisfies the constraints that are defined on the NEWEMP table.

Example: Consider this UPDATE statement:

```
UPDATE NEWEMP
  SET DEPT = '166'
  WHERE FIRSTNME = 'MARY' AND LASTNAME= 'SMITH';
```

This update fails because the value of DEPT is '166,' which violates the check constraint on NEWEMP that DEPT values must be between '001' and '100.'

GUPI

Db2 indexes

An *index* is an ordered set of pointers to rows of a table. Db2 can use indexes to improve performance and ensure uniqueness. Understanding the structure of Db2 indexes can help you achieve the best performance for your system.

Conceptually, you can think of an index to the rows of a Db2 table like you think of an index to the pages of a book. Each index is based on the values of data in one or more columns of a table.

The main purpose of an index is to improve performance for access to the data. In most cases, access to data is faster with an index than with a scan of the data. For example, you can create an index on the DEPTNO column of the sample DEPT table so that Db2 can easily locate a specific department and avoid reading through each row of, or *scanning*, the table.

An index is stored separately from the data in the table. Each index is physically stored in its own index space. When you define an index by using the CREATE INDEX statement, Db2 builds this structure and maintains it automatically. However, you can perform necessary maintenance such as reorganizing it or recovering the index.

Another purpose of an index is to ensure uniqueness. For example, a unique index on the employee table ensures that no two employees have the same employee number.

In most cases, the users of an index are unaware that it is being used because Db2 decides whether to use the index to access the table.

Db2 supports simple indexes and extended indexes. An extended index is one of the following objects:

- An expression-based index
- A spatial index
- An XML index

Be aware that indexes have both benefits and disadvantages. A greater number of indexes can simultaneously improve the access performance of a particular transaction and require additional processing for inserting, updating, and deleting index keys.

Creation of indexes
Indexes provide efficient access to table data, but can require additional processing when you modify data in a table.

CREATE INDEX (Db2 SQL)

Db2 views

A *view* is an alternative way of representing data that exists in one or more tables. A view can include all or some of the columns from one or more base tables.

Introductory concepts

"Creation of views" on page 117
"Views on the sample tables" on page 332

A view is a named specification of a result table. Conceptually, creating a view is somewhat like using binoculars. You might look through binoculars to see an entire landscape or to look at a specific image within the landscape, such as a tree.

You can create a view that:

- Combines data from different base tables
- Is based on other views or on a combination of views and tables
- Omits certain data, thereby shielding some table data from users

In fact, these are common underlying reasons to use a view. Combining information from base tables and views simplifies retrieving data for a user, and limiting the data that a user can see is useful for security. You can use views for a number of different purposes. A view can:

- Control access to a table
- Make data easier to use
- Simplify authorization by granting access to a view without granting access to the table
- Show only portions of data in the table
- Show summary data for a given table
- Combine two or more tables in meaningful ways
- Show only the selected rows that are pertinent to the process that uses the view

To define a view, you use the CREATE VIEW statement and assign a name (up to 128 characters in length) to the view. Specifying the view in other SQL statements is effectively like running an SQL SELECT statement. At any time, the view consists of the rows that would result from the SELECT statement that it contains. You can think of a view as having columns and rows just like the base table on which the view is defined.

You also can specify a period specification for a view, subject to certain restrictions.

Examples

GUPI

Example 1

Example 1: The following figure shows a view of the EMP table that omits sensitive employee information and renames some of the columns.

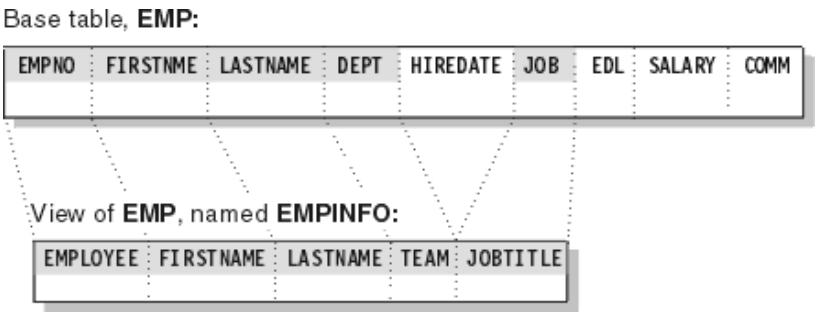


Figure 5. A view of the EMP table

Figure note: The EMPINFO view represents a table that includes columns named EMPLOYEE, FIRSTNAME, LASTNAME, TEAM, and JOBTITLE. The data in the view comes from the columns EMPNO, FIRSTNME, LASTNAME, DEPT, and JOB of the EMP table.

Example 2

The following CREATE VIEW statement defines the EMPINFO view that is shown in the preceding figure:

```
CREATE VIEW EMPINFO (EMPLOYEE, FIRSTNAME, LASTNAME, TEAM, JOBTITLE)
  AS SELECT EMPNO, FIRSTNME, LASTNAME, DEPT, JOB
  FROM EMP;
```

When you define a view, Db2 stores the definition of the view in the Db2 catalog. However, Db2 does not store any data for the view itself, because the data exists in the base table or tables.

Example 3

You can narrow the scope of the EMPINFO view by limiting the content to a subset of rows and columns that includes departments A00 and C01 only:

```
CREATE VIEW EMPINFO (EMPLOYEE, FIRSTNAME, LASTNAME, TEAM, JOBTITLE)
  AS SELECT EMPNO, FIRSTNME, LASTNAME, WORKDEPT, JOB
  FROM EMP
  WHERE WORKDEPT = 'A00' OR WORKDEPT = 'C01';
```

GUPI

In general, a view inherits the attributes of the object from which it is derived. Columns that are added to the tables after the view is defined on those tables do not appear in the view.

Restriction: You cannot create an index for a view. In addition, you cannot create any form of a key or a constraint (referential or otherwise) on a view. Such indexes, keys, or constraints must be built on the tables that the view references.

To retrieve or access information from a view, you use views like you use base tables. You can use a SELECT statement to show the information from the view. The SELECT statement can name other views and tables, and it can use the WHERE, GROUP BY, and HAVING clauses. It cannot use the ORDER BY clause or name a host variable.

Whether a view can be used in an insert, update, or delete operation depends on its definition. For example, if a view includes a foreign key of its base table, INSERT and UPDATE operations that use the view are subject to the same referential constraint as the base table. Likewise, if the base table of a view is a parent table, DELETE operations that use the view are subject to the same rules as DELETE operations on the base table. *Read-only* views cannot be used for insert, update, and delete operations.

Related information

Implementing Db2 views (Db2 Administration Guide)

Employee table (DSN8B10.EMP)
The sample employee table identifies all employees by an employee number and lists basic personnel information.

CREATE VIEW (Db2 SQL)

Aliases

An *alias* is a substitute for the three-part name of a table or view.

An alias can be defined at a local server and can refer to a table or view that is at the current server or a remote server. The alias name can be used wherever the table name or view name can be used to refer to the table or view in an SQL statement.

Suppose that data is occasionally moved from one Db2 subsystem to another. Ideally, users who query that data are not affected when this activity occurs. They always want to log on to the same system and access the same table or view, regardless of where the data resides. You can achieve this result by using an alias for an object name.

An alias can be a maximum of 128 characters, qualified by an owner ID. You use the CREATE ALIAS and DROP ALIAS statements to manage aliases.

GUPI

Note: Assume that you create an alias as follows:

```
CREATE ALIAS TESTTAB FOR USIBMSTODB22.IDEMP01.EMP;
```

If a user with the ID JONES dynamically creates the alias, JONES owns the alias, and you query the table like this:

```
SELECT SUM(SALARY), SUM(BONUS), SUM(COMM)
  FROM JONES.TESTTAB;
```

GUPI

The object for which you are defining an alias does not need to exist when you execute the CREATE ALIAS statement. However, the object must exist when a statement that refers to the alias executes.

When you want an application to access a server other than the server that is specified by a location name, you do not need to change the location name. Instead, you can use a location alias to override the location name that an application uses to access a server. As a result, a Db2 for z/OS requester can

access multiple Db2 databases that have the same name but different network addresses. Location aliases allow easier migration to a Db2 server and minimize application changes.

After you create an alias, anyone who has authority over the object that the alias is referencing can use that alias. A user does not need a separate privilege to use the alias.

Related reference

CREATE ALIAS (Db2 SQL)

Triggers

A *trigger* defines a set of actions that are executed when a delete, insert, or update operation occurs on a specified table or view. When such an operation is executed, the trigger is said to be *activated*.

You can use triggers along with referential constraints and check constraints to enforce data integrity rules. Triggers are more powerful than constraints because you can use them to do the following things:

- Update other tables
- Automatically generate or transform values for inserted or updated rows
- Invoke functions that perform operations both inside and outside of Db2

For example, assume that you need to prevent an update to a column when a new value exceeds a certain amount. Instead of preventing the update, you can use a trigger. The trigger can substitute a valid value and invoke a procedure that sends a notice to an administrator about the attempted invalid update.

You define triggers with the CREATE TRIGGER statement.

Triggers move the business rule application logic into the database, which results in faster application development and easier maintenance. The business rule is no longer repeated in several applications, and the rule is centralized to the trigger. For example, Db2 can check the validity of the changes that any application makes to a salary column, and you are not required to change application programs when the logic changes.

There are a number of criteria that are defined when creating a trigger, which are used to determine when a trigger should be activated.

- The *subject table* (also known as the triggering table) defines the table or view for which the trigger is defined.
- The *trigger event* defines a specific SQL operation that modifies the subject table. The operation could be a delete, insert, or update.
- The *trigger activation time* defines whether the trigger should be activated before or after the trigger event is performed on the subject table.

The statement that causes a trigger to be activated includes a set of *affected rows*. These are the rows of the subject table that are being deleted, inserted or updated. The *trigger granularity* defines whether the actions of the trigger are to be performed once for the statement, or once for each of the rows in the set of affected rows.

The *trigger action* consists of an optional search condition and a set of SQL statements that are executed whenever the trigger is activated. The SQL statements are only executed if no search condition is specified, or the specified search condition evaluates to true.

The triggered action can refer to the values in the set of affected rows. This is supported through the use of *transition variables*. Transition variables use the names of the columns in the subject table, qualified by a specified name that identifies whether the reference is to the old value (prior to the update) or the new value (after the update). The new value can also be changed using an assignment in a before update or insert trigger.

Another means of referring to the values in the set of affected rows is by using *transition tables*. Transition tables also use the names of the columns of the subject table, but have a name specified that allows the complete set of affected rows to be treated as a table. Transition tables can only be used in after triggers, and cannot be modified. Separate transition tables can be defined for old and new values.

Multiple triggers can be specified for a combination of table, event, or activation time. The order in which the triggers are activated is the same as the order in which they were created. Thus, the most recently created trigger is the last trigger that is activated.

The activation of a trigger might cause *trigger cascading*. This is the result of the activation of one trigger that executes SQL statements that cause the activation of other triggers, or even the same trigger again. The triggered actions might also cause updates as a result of the original modification, which might result in the activation of additional triggers. With trigger cascading, a significant chain of triggers might be activated, causing significant change to the database as a result of a single delete, insert or update statement.

The actions that are performed in the trigger are considered to be part of the operation that caused the trigger to be executed.

- The database manager ensures that the operation and the triggers that are executed as a result of that operation either all complete or are all backed out. Operations that occurred prior to the triggering operation are not affected.
- The database manager effectively checks all constraints (except for a constraint with a RESTRICT delete rule) after the operation and the associated triggers have been executed.

Creation of triggers
You can use triggers to define and enforce business rules that involve different states of the data. Triggers automatically execute a set of SQL statements whenever a specified event occurs. These statements validate and edit database changes, read and modify the database, and invoke functions that perform various operations.

User-defined types

A *user-defined data type* is a data type that is defined to the database using a CREATE statement.

A user-defined data type is a distinct type or an array type.

A *distinct type* is a user-defined type that shares its internal representation with a built-in data type (its source type), but is considered to be a separate and incompatible data type for most operations. A distinct type is created with an SQL CREATE TYPE (distinct) statement. A distinct type can be used to define a column of a table, or a parameter of a routine.

An *array type* is a user-defined data type that consists of an ordered set of elements of a single built-in data type. Elements can be accessed and modified by their index position. An array type is created with an SQL CREATE TYPE (array) statement. An array type can be used as a parameter of an SQL routine and as a variable in an SQL routine.

Related concepts
Array types (Db2 SQL)
Distinct types (Db2 SQL)
Related reference
CREATE TYPE (Db2 SQL)

Routines

A *routine* is an executable SQL object. The two types of routines are functions and stored procedures.

Functions

A *function* is a routine that can be invoked from within other SQL statements and that returns a value or a table.

Functions are classified as either SQL functions or external functions. SQL functions are written using SQL statements, including SQL procedural language (SQL PL). External functions reference a host language program. The host language program can contain SQL, but does not require SQL.

You define functions by using the CREATE FUNCTION statement. You can classify functions as built-in functions, user-defined functions, or cast functions that are generated for distinct types. Functions can also be classified as aggregate, scalar, or table functions, depending on the input data values, result values, and the context in which they can be invoked.

Related tasks

Creating a user-defined function (Db2 Application programming and SQL)

Creation of user-defined functions
You can create your own functions in Db2 to simplify your queries.

Built-in functions (Db2 SQL)

Functions (Db2 SQL)

Types of functions (Db2 SQL)

Procedures

A *procedure*, also known as a stored procedure, is a routine that you can call to perform operations that can include SQL statements.

Procedures are classified as either SQL procedures or external procedures. SQL procedures contain only SQL statements, including SQL procedural language (SQL PL). External procedures reference a host language program that might or might not contain SQL statements.

Db2 for z/OS supports the following types of procedures:

Native SQL procedures

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is contained and specified in the procedure definition along with various attributes of the procedure. A package is generated for a native SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2. Each time that the procedure is invoked, the package executes one or more times.

All SQL procedures that are created with a CREATE PROCEDURE statement that does not specify the FENCED or EXTERNAL options are native SQL procedures. More capabilities are supported for native SQL procedures, they usually perform better than external SQL procedures, and no associated C program is generated for them.

For more information, see Creating native SQL procedures (Db2 Application programming and SQL).

External stored procedures

The procedure body is an external program that is written in a programming language such as C, C++, COBOL, or Java and it can contain SQL statements. The source code for an external stored procedure is separate from the procedure definition and is bound into a package. The name of the external executable is specified as part of the procedure definition along with various attributes of the procedure. All programs must be designed to run using Language Environment. Your COBOL and C++ stored procedures can contain object-oriented extensions. Each time that the stored procedure is invoked, the logic in the procedure controls whether the package executes and how many times.

For more information, see Creating external stored procedures (Db2 Application programming and SQL).

External SQL procedures (deprecated)

The procedure body is written exclusively in SQL statements, including SQL procedural language (SQL PL) statements. The procedure body is specified in the procedure definition along with various attributes of the procedure. A C program and an associated package are generated for an external SQL procedure. It contains the procedure body, including control statements. It might sometimes also include statements generated by Db2.Each time that the procedure is invoked, the package executes one or more times.

Native SQL procedures are more fully supported, easier to maintain, and typically perform better than external SQL procedures, which are deprecated.

For more information, see Creating external SQL procedures (deprecated) (Db2 Application programming and SQL).

SQL control statements are supported in SQL procedures. Control statements 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 procedures provide the same benefits as procedures in a host language. That is, a common piece of code needs to be written and maintained only once and can be called from several programs.

SQL procedures provide additional benefits when they contain SQL statements. In this case, SQL procedures can reduce or eliminate network delays that are associated with communication between the client and server and between each SQL statement. SQL procedures can improve security by providing a user the ability to invoke only a procedure instead of providing them with the ability to execute the SQL that the procedure contains.

You define procedures by using the CREATE PROCEDURE statement.

Related reference

Procedures that are supplied with Db2 (Db2 SQL)

Use of an application program as a stored procedure
A stored procedure is a compiled program that can execute SQL statements.

External stored procedures (Db2 Application programming and SQL)

SQL control statements for external SQL procedures (Db2 SQL)

SQL control statements for SQL routines (Db2 SQL)

Implementing Db2 stored procedures ()

Procedures that are supplied with Db2 (Db2 SQL)

Sequences

A *sequence* is a stored object that simply generates a sequence of numbers in a monotonically ascending (or descending) order. A sequence provides a way to have Db2 automatically generate unique integer primary keys and to coordinate keys across multiple rows and tables.

A sequence can be used to exploit parallelization, instead of programmatically generating unique numbers by locking the most recently used value and then incrementing it.

Sequences are ideally suited to the task of generating unique key values. One sequence can be used for many tables, or a separate sequence can be created for each table requiring generated keys. A sequence has the following properties:

- Guaranteed, unique values, assuming that the sequence is not reset and does not allow the values to cycle
- Monotonically increasing or decreasing values within a defined range
- Can increment with a value other than 1 between consecutive values (the default is 1).
- Recoverable. If Db2 should fail, the sequence is reconstructed from the logs so that Db2 guarantees that unique sequence values continue to be generated across a Db2 failure.

Values for a given sequence are automatically generated by Db2. Use of Db2 sequences avoids the performance bottleneck that results when an application implements sequences outside the database. The counter for the sequence is incremented (or decremented) independently of the transaction. In some cases, gaps can be introduced in a sequence. A gap can occur when a given transaction increments a sequence two times. The transaction might see a gap in the two numbers that are generated because there can be other transactions concurrently incrementing the same sequence. A user might not realize that other users are drawing from the same sequence. Furthermore, it is possible that a given sequence can appear to have generated gaps in the numbers, because a transaction that might have generated a sequence number might have rolled back or the Db2 subsystem might have failed. Updating a sequence is not part of a transaction's unit of recovery.

A sequence is created with a CREATE SEQUENCE statement. A sequence can be referenced using a *sequence-reference*. A sequence reference can appear most places that an expression can appear. A

sequence reference can specify whether the value to be returned is a newly generated value, or the previously generated value.

Although there are similarities, a sequence is different than an identity column. A sequence is an object, whereas an identity column is a part of a table. A sequence can be used with multiple tables, but an identity column is tied to a single table.

CREATE SEQUENCE (Db2 SQL)

Db2 system objects

Unlike the Db2 data structures that users create and access, Db2 controls and accesses system objects.

Db2 has a comprehensive infrastructure that enables it to provide data integrity, performance, and the ability to recover user data. In addition, Parallel Sysplex® data sharing uses shared system objects.

Related concepts

Db2 database objects overview
In Db2 for z/OS, you use database objects, such as tables, table spaces, indexes, index spaces, keys, views, and databases to organize and access your data.

Db2 catalog

Db2 maintains a set of tables that contain information about the data that Db2 controls. These tables are collectively known as the *catalog*.

The catalog tables contain information about Db2 objects such as tables, views, and indexes. When you create, alter, or drop an object, Db2 inserts, updates, or deletes rows of the catalog that describe the object.

The Db2 catalog consists of tables of data about everything defined to the Db2 system, including table spaces, indexes, tables, copies of table spaces and indexes, and storage groups. The system database DSNDB06 contains the Db2 catalog.

When you create, alter, or drop any structure, Db2 inserts, updates, or deletes rows of the catalog that describe the structure and tell how the structure relates to other structures. For example, SYSIBM.SYSTABLES is one catalog table that records information when a table is created. Db2 inserts a row into SYSIBM.SYSTABLES that includes the table name, its owner, its creator, and the name of its table space and its database.

To understand the role of the catalog, consider what happens when the EMP table is created. Db2 records the following data:

Table information
To record the table name and the name of its owner, its creator, its type, the name of its table space, and the name of its database, Db2 inserts a row into the catalog.

Column information
To record information about each column of the table, Db2 inserts the name of the table to which the column belongs, its length, its data type, and its sequence number by inserting a row into the catalog for each column of the table.

Authorization information
To record that the owner of the table has authorization to create the table, Db2 inserts a row into the catalog.

Tables in the catalog are like any other database tables with respect to retrieval. If you have authorization, you can use SQL statements to look at data in the catalog tables in the same way that you retrieve data from any other table in the Db2 database. Db2 ensures that the catalog contains accurate object descriptions. If you are authorized to access the specific tables or views on the catalog, you can use SELECT on the catalog. You can use INSERT, UPDATE, and DELETE statements on updatable catalog tables, but you cannot use TRUNCATE or MERGE on the catalog.

The *communications database* (CDB) is part of the Db2 catalog. The CDB consists of a set of tables that establish conversations with remote database management systems (DBMSs). The distributed data facility (DDF) uses the CDB to send and receive distributed data requests.

Tip: For best results, check the consistency of the Db2 catalog and directory regularly, even outside of the migration process. For detailed instructions, see Verify the integrity of Db2 table spaces and Check for consistency between catalog tables.

Db2 catalog tables (Db2 SQL)

Db2 directory

The Db2 directory contains information that Db2 uses during normal operation.

You can use SQL to retrieve data only from the directory tables that are listed in Db2 directory tables (Db2 SQL). You can submit queries against the Db2 catalog for Db2 directory objects.

The directory consists of a set of Db2 tables that are stored in table spaces in system database DSNDB01. Each of the table spaces that are listed in the following table is contained in a VSAM linear data set.

Table 8. Directory table spaces

Table space name	Description
SCT02	Contains the internal form of SQL statements that are contained in an application. If you bound a plan with SQL statements in a prior release, Db2 created a structure in SCT02.
SPT01 Skeleton package	Contains the internal form of SQL statements that are contained in a package.
SYSSPUXA	Contains the contents of a package selection.
SYSSPUXB	Contains the contents of a package explain block.
SYSLGRNX Log range	Tracks the opening and closing of table spaces, indexes, or partitions. By tracking this information and associating it with relative byte addresses (RBAs) as contained in the Db2 log, Db2 can reduce recovery time by reducing the amount of log that must be scanned for a particular table space, index, or partition.
SYSUTILX System utilities	Contains a row for every utility job that is running. The row persists until the utility is finished. If the utility terminates without completing, Db2 uses the information in the row when you restart the utility.
DBD01 Database descriptor (DBD)	Contains internal information, called *database descriptors* (DBDs), about the databases that exist within the Db2 subsystem. Each database has exactly one corresponding DBD that describes the database, table spaces, tables, table check constraints, indexes, and referential relationships. A DBD also contains other information about accessing tables in the database. Db2 creates and updates DBDs whenever their corresponding databases are created or updated.
SYSDBDXA	Contains the contents of a DBD section.

Tip: For best results, check the consistency of the Db2 catalog and directory regularly, even outside of the migration process. For detailed instructions, see Verify the integrity of Db2 table spaces and Check for consistency between catalog tables.

Active and archive logs

Db2 records all data changes and other significant events in a log.

If you keep these logs, Db2 can re-create those changes for you in the event of a failure or roll the changes back to a previous point in time.

Db2 writes each log record to a disk data set called the *active log*. When the active log is full, Db2 copies the contents of the active log to a disk or magnetic tape data set called the *archive log*.

You can choose either single logging or dual logging.

- A single active log contains up to 93 active log data sets.
- With dual logging, the active log has twice the capacity for active log data sets, because two identical copies of the log records are kept.

Each Db2 subsystem manages multiple active logs and archive logs. The following facts are true about each Db2 active log:

- Each log can be duplexed to ensure high availability.
- Each active log data set is a VSAM linear data set (LDS).
- Db2 supports striped active log data sets.

Related tasks

Managing the log and the bootstrap data set (Db2 Administration Guide)

Improving Db2 log performance (Db2 Performance)

Related information

Reading log records (Db2 Administration Guide)

Bootstrap data set

The *bootstrap data set* (*BSDS*) is a VSAM key-sequenced data set (KSDS). This KSDS contains information that is critical to Db2, such as the names of the logs. Db2 uses information in the BSDS for system restarts and for any activity that requires reading the log.

Specifically, the BSDS contains:

- An inventory of all active and archive log data sets that are known to Db2. Db2 uses this information to track the active and archive log data sets. Db2 also uses this information to locate log records to satisfy log read requests during normal Db2 system activity and during restart and recovery processing.
- A wrap-around inventory of all recent Db2 checkpoint activity. Db2 uses this information during restart processing.
- The distributed data facility (DDF) communication record, which contains information that is necessary to use Db2 as a distributed server or requester.
- Information about buffer pools.

Because the BSDS is essential to recovery in the event of subsystem failure, during installation Db2 automatically creates two copies of the BSDS and, if space permits, places them on separate volumes.

The BSDS can be duplexed to ensure availability.

At migration to Db2 12, you cannot start Db2 12 until the BSDS is converted to use the 10-byte RBA and LRSN formats. You can convert the BSDS before or during the Db2 12 migration process.

For instructions, see Converting the BSDS to the 10-byte RBA and LRSN (Db2 Administration Guide).

Related tasks

Managing the log and the bootstrap data set (Db2 Administration Guide)

Convert the BSDS, Db2 catalog, and directory to 10-byte RBA and LRSN format (Optional) (Db2 Installation and Migration)

Buffer pools

Buffer pools are areas of virtual storage that temporarily store pages of table spaces or indexes.

When an application program accesses a row of a table, Db2 places the page that contains that row in a buffer. Access to data in this temporary storage is faster than accessing data on a disk. If the required data is already in a buffer, the application program does not need to wait for it to be retrieved from disk, so the time and cost of retrieving the page is reduced.

Buffer pools require monitoring and tuning. Buffer pool sizes are critical to the performance characteristics of an application or group of applications that access data in those buffer pools.

You can specify default buffer pools for user data and for indexes. A special type of buffer pool that is used only in Parallel Sysplex data sharing is the *group buffer pool*, which resides in the coupling facility. Group buffer pools reside in a special PR/SM LPAR logical partition called a *coupling facility*, which enables several Db2 subsystems to share information and control the coherency of data.

Buffer pools reside in the DBM1 address space. The maximum size of a buffer pool is 16 TB.

Related concepts

The role of buffer pools in caching data

Buffer pools are a key element of Db2 performance, and help you to avoid delays when retrieving data.

Related tasks

Tuning database buffer pools (Db2 Performance)

Calculating buffer pool size (Db2 Installation and Migration)

Enabling automatic buffer pool size management (Db2 Performance)

Data definition control support database

The *data definition control support* (DDCS) database refers to a user-maintained collection of tables that are used by data definition control support to restrict the submission of specific Db2 DDL (data definition language) statements to selected application identifiers (plans or collections of packages).

This database is automatically created during installation. After this database is created, you must populate the tables to use this facility. The system name for this database is DSNRGFDB.

The resource limit facility

The *resource limit facility* (sometimes abbreviated RLF) enables you to control the amount of processor resources that are used by SQL statements.

GUPI

Resource limits apply to the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

You can establish a single limit for all users, different limits for individual users, or both. You can choose to have these limits applied before the statement is executed through *predictive governing*, or while a

statement is running, through *reactive governing*. You can also use reactive and predictive governing in combination. You define these limits in one or more *resource limit tables*, named DSNRLST*xx* or DSNRLMT*xx*, depending on the monitoring purpose. GUPI

Related concepts

Resource limit facility controls (Db2 Performance)

Related tasks

Setting limits for system resource usage by using the resource limit facility (Db2 Performance)

Related reference

Resource limit facility tables (Db2 Performance)

-START RLIMIT (Db2) (Db2 Commands)

Work file database

Use the *work file database* as storage for processing SQL statements that require working space, such as that required for a sort.

The work file database is used as storage for Db2 work files for processing SQL statements that require working space (such as the space that is required for a sort), and as storage for created global temporary tables and declared global temporary tables.

Db2 creates a work file database and some table spaces in it for you at installation time. You can create additional work file table spaces at any time. You can drop, re-create, and alter the work file database or the table spaces in it, or both, at any time.

In a non-data-sharing environment, the work file database is named DSNDB07. In a data sharing environment, each Db2 member in the data sharing group has its own work file database.

You can also use the work file database for all temporary tables.

Application processes and transactions

An application process involves running one or more programs. Different application processes might involve running different programs or running the same program at different times. When an application interacts with a Db2 database, a transaction begins.

Many different types of programs access Db2 data: user-written applications, SQL statements that users enter dynamically, and even utilities. The single term that describes any type of access to Db2 data is called an *application process*. All SQL programs run as part of an application process.

A *transaction* is a sequence of actions between the application and the database; the sequence begins when data in the database is read or written. A transaction is also known as a *unit of work*.

Example: Consider what happens when you access funds in a bank account. A banking transaction might involve the transfer of funds from one account to another. During the transaction, an application program first subtracts the funds from the first account, and then it adds the funds to the second account. Following the subtraction step, the data is inconsistent. Consistency is reestablished after the funds are added to the second account.

To ensure data consistency, Db2 uses a variety of techniques that include a commit operation, a rollback operation, and locking.

When the subtraction and addition steps of the banking transaction are complete, the application can use the commit operation to end the transaction, thereby making the changes available to other application processes. The *commit* operation makes the database changes permanent.

Consider what happens if more than one application process requests access to the same data at the same time. Or, under certain circumstances, an SQL statement might run concurrently with a utility on the same table space. Db2 uses locks to maintain data integrity under these conditions to prevent, for example, two application processes from updating the same row of data simultaneously.

Db2 acquires locks to prevent uncommitted changes that are made by one application process from being perceived by any other. Db2 automatically releases all locks that it has acquired on behalf of an

application process when that process ends, but an application process can also explicitly request that locks be released sooner. A commit operation releases locks that an application process has acquired and commits database changes that were made by the same process.

Db2 also provides a way to *back out* uncommitted changes that an application process makes. A back out might be necessary in the event of a failure on the part of an application process or in a *deadlock* situation. Deadlock occurs when contention for the use of a resource, such as a table, cannot be resolved. An application process, however, can explicitly request that its database changes be backed out. This operation is called *rollback*. The interface that an SQL program uses to explicitly specify these commit and rollback operations depends on the environment. For example, in the JDBC environment, applications use commit and rollback methods to commit or roll back transactions.

Related concepts

Programming for Db2 for z/OS

You use application programs that contain SQL statements to perform data definition or data manipulation operations on Db2 for z/OS objects.

Application processes, concurrency, and recovery

All SQL programs execute as part of an *application process*. An application process involves the execution of one or more programs, and it is the unit to which Db2 allocates resources and locks.

Different application processes might involve the execution of different programs, or different executions of the same program. The means of initiating and terminating an application process are dependent on the environment.

Locking, commit, and rollback

More than one application process might request access to the same data at the same time. Furthermore, under certain circumstances, an SQL statement can execute concurrently with a utility on the same table space. *Locking* is used to maintain data integrity under such conditions, preventing, for example, two application processes from updating the same row of data simultaneously.

Db2 implicitly acquires locks to prevent uncommitted changes made by one application process from being perceived by any other. Db2 will implicitly release all locks it has acquired on behalf of an application process when that process ends, but an application process can also explicitly request that locks be released sooner. A *commit* operation releases locks acquired by the application process and commits database changes made by the same process.

Db2 provides a way to *back out* uncommitted changes made by an application process. This might be necessary in the event of a failure on the part of an application process, or in a *deadlock* situation. An application process, however, can explicitly request that its database changes be backed out. This operation is called *rollback*.

The interface used by an SQL program to explicitly specify these commit and rollback operations depends on the environment. If the environment can include recoverable resources other than Db2 databases, the SQL COMMIT and ROLLBACK statements cannot be used. Thus, these statements cannot be used in an IMS, CICS, or WebSphere environment.

Unit of work

A *unit of work* is a recoverable sequence of operations within an application process. A unit of work is sometimes called a *logical unit of work*.

At any time, an application process has a single unit of work, but the life of an application process can involve many units of work as a result of commit or full rollback operations.

A unit of work is initiated when an application process is initiated. A unit of work is also initiated when the previous unit of work is ended by something other than the end of the application process. A unit of work is ended by a commit operation, a full rollback operation, or the end of an application process. A commit or rollback operation affects only the database changes made within the unit of work it ends. While these changes remain uncommitted, other application processes are unable to perceive them unless they are running with an isolation level of uncommitted read. The changes can still be backed out. Once committed, these database changes are accessible by other application processes and can no longer be

backed out by a rollback. Locks acquired by Db2 on behalf of an application process that protects uncommitted data are held at least until the end of a unit of work.

The initiation and termination of a unit of work define *points of consistency* within an application process. A point of consistency is a claim by the application that the data is consistent. For example, a banking transaction might involve the transfer of funds from one account to another. Such a transaction would require that these funds be subtracted from the first account, and added to the second. Following the subtraction step, the data is inconsistent. Only after the funds have been added to the second account is consistency reestablished. When both steps are complete, the commit operation can be used to end the unit of work, thereby making the changes available to other application processes. The following figure illustrates this concept.

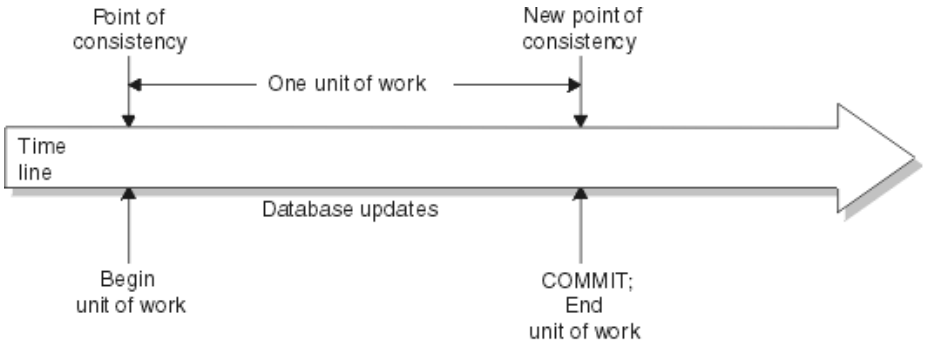


Figure 6. Unit of work with a commit operation

Unit of recovery

A *Db2 unit of recovery* is a recoverable sequence of operations executed by Db2 for an application process.

If a unit of work involves changes to other recoverable resources, the unit of work will be supported by other units of recovery. If relational databases are the only recoverable resources used by the application process, then the scope of the unit of work and the unit of recovery are the same and either term can be used.

Rolling back work

Db2 can back out all changes made in a unit of recovery or only selected changes. Only backing out all changes results in a point of consistency.

Rolling back all changes

The SQL ROLLBACK statement without the TO SAVEPOINT clause specified causes a full rollback operation. If such a rollback operation is successfully executed, Db2 backs out uncommitted changes to restore the data consistency that existed when the unit of work was initiated.

That is, Db2 undoes the work, as shown in the following figure:

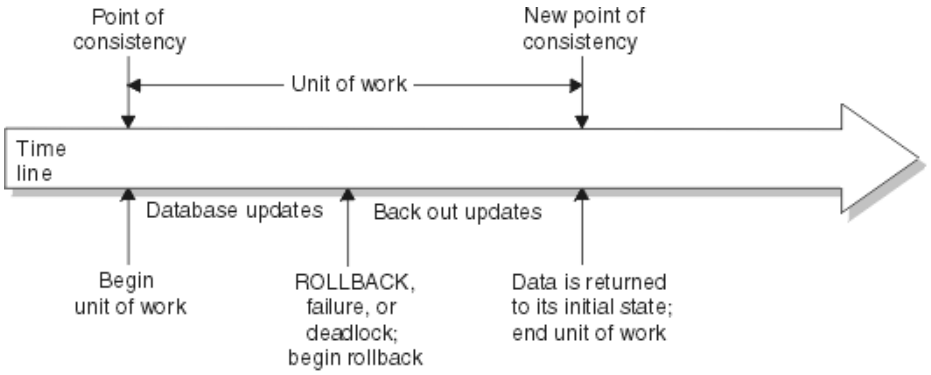


Figure 7. Rolling back all changes from a unit of work

Rolling back selected changes using savepoints

A *savepoint* represents the state of data at some particular time during a unit of work. An application process can set savepoints within a unit of work, and then as logic dictates, roll back only the changes that were made after a savepoint was set.

For example, part of a reservation transaction might involve booking an airline flight and then a hotel room. If a flight gets reserved but a hotel room cannot be reserved, the application process might want to undo the flight reservation without undoing any database changes made in the transaction prior to making the flight reservation. SQL programs can use the SQL SAVEPOINT statement to set savepoints, the SQL ROLLBACK statement with the TO SAVEPOINT clause to undo changes to a specific savepoint or the last savepoint that was set, and the SQL RELEASE SAVEPOINT statement to delete a savepoint. The following figure illustrates this concept.

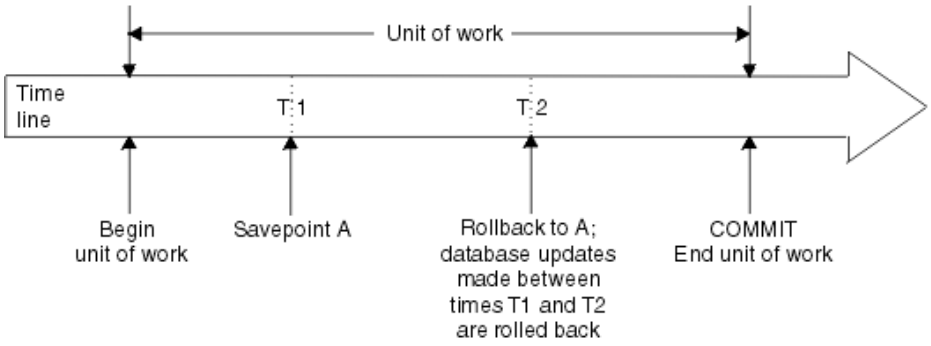


Figure 8. Rolling back changes to a savepoint within a unit of work

Packages and application plans

A *package* contains control structures that Db2 uses when it runs SQL statements. An *application plan* relates an application process to a local instance of Db2 and specifies processing options.

Packages are produced during program preparation. You can think of the control structures as the bound or operational form of SQL statements. All control structures in a package are derived from the SQL statements that are embedded in a single source program.

An application plan contains a list of package names.

Db2 applications require an application plan. Packages make application programs more flexible and easier to maintain.

In general, you create plans and packages by using the Db2 commands BIND PLAN and BIND PACKAGE.

Example

The following figure shows an application plan that contains two packages. Suppose that you decide to change the SELECT statement in package AA to select data from a different table. In this case, you need to bind only package AA again and not package AB.

GUPI

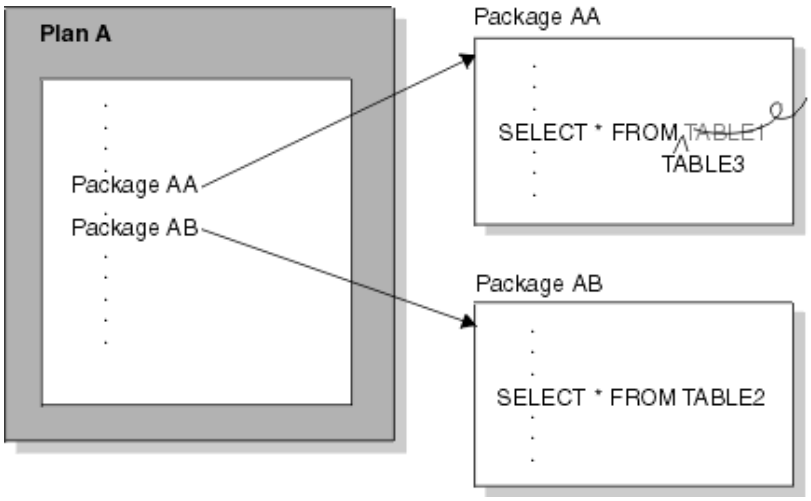


Figure 9. Application plan and packages

GUPI

Packages for triggers and routines

A *trigger package* is a special type of package that is created when you issue a CREATE TRIGGER statement. It contains the trigger body and the WHEN clause for the trigger. When the trigger activates, the package executes one or more times.

Packages are also generated, or explicitly created for certain SQL routines, including stored procedures and certain types of user defined functions. The package for a routine contains the routine body, including control statements. The package might also contain statements that are generated by Db2. Each time that the routine is invoked, the package executes one or more times.

Specifically, Db2 generates a package automatically for the following types of SQL routines:

- External scalar functions
- External table functions
- Compiled SQL scalar functions
- Native SQL procedures
- External SQL procedures

For external stored procedures, you explicitly bind the source code for the stored procedure into a package before or you issue the CREATE PROCEDURE statement. Each time that the external procedure is invoked, the package executes one or more times.

Packages for JDBC, SQLJ, and ODBC applications

Packages for JDBC, SQLJ, and ODBC applications serve different purposes. For more information, see and Db2 ODBC run time environment setup (Db2 Programming for ODBC).

Related concepts

Programming for Db2 for z/OS
You use application programs that contain SQL statements to perform data definition or data manipulation operations on Db2 for z/OS objects.

Preparation process for an application program
How you prepare an application program to run depends on the type of application. The program preparation steps for applications vary based on the type of programming language that is used.

Related reference

SET CURRENT PACKAGE PATH (Db2 SQL)
SET CURRENT PACKAGESET (Db2 SQL)

CREATE FUNCTION (Db2 SQL)
CREATE PROCEDURE (Db2 SQL)
CREATE TRIGGER (Db2 SQL)

Subsystem parameters

Subsystem parameters are settings that apply to a Db2 for z/OS subsystem. For example, the AUTH subsystem parameter controls whether Db2 checks authorizations. Many users colloquially refer to subsystem parameters as *zparms*.

Generally, system administrators set the value of subsystem parameters when they install or migrate Db2. After installation or migration, a system administrator can change the values of subsystem parameters if needed. However, because these parameters can affect the entire subsystem, system administrators should first carefully evaluate and plan for the impact of any changed values. In some cases, changing the value of a subsystem parameter requires Db2 to be stopped and restarted.

Most subsystem parameters are listed on installation panels. However, some parameters are not. For a list of subsystem parameters, including those parameters that are not on panels, see Directory of subsystem parameters, panel fields, and application default values (Db2 Installation and Migration). This list includes important characteristics of each parameter, including whether changing a the parameter requires you to stop and restart Db2.

Subsystem parameter values are contained in a load module that is generated by installation job DSNTIJUZ. The name of this subsystem parameters load module is specified in the PARAMETER MODULE field of installation panel DSNTIPO3. The default name is DSNZPARM. However, to avoid naming conflicts with other members, you should use a different name. Sometimes the naming convention DSNZP*xxx* is used. In a data sharing environment, each member must have its own subsystem parameter load module.

One way to check the subsystem parameter values is to call stored procedure SYSPROC.ADMIN_INFO_SYSPARM by running sample job DSNTEJ6Z.

Related concepts

Subsystem parameters and application defaults in a data sharing environment (Db2 Installation and Migration)

Application programming default values
Each Db2 for z/OS subsystem has a set of *application programming default values*. These values are the default settings for SQL applications that run on that subsystem. For example, the ENSCHEME application programming default value specifies the default encoding scheme for new objects. Some users colloquially refer to these values as *DECP values*.

The Db2 installation CLIST panel session (Db2 Installation and Migration)

Related reference

PARAMETER MODULE field (Db2 Installation and Migration)

Storage structures

In Db2, a *storage structure* is a set of one or more VSAM data sets that hold Db2 tables or indexes. A storage structure is also called a *page set*.

The two primary types of storage structures in Db2 for z/OS are table spaces and index spaces.

Related concepts

Db2 database objects overview
In Db2 for z/OS, you use database objects, such as tables, table spaces, indexes, index spaces, keys, views, and databases to organize and access your data.

Related information

Implementing Db2 table spaces (Db2 Administration Guide)

Implementing Db2 indexes (Db2 Administration Guide)

Db2 databases

Db2 *databases* are a set of Db2 structures that include a collection of tables, their associated indexes, and the table spaces in which they reside. You define a database by using the CREATE DATABASE statement.

Whenever a table space is created, it is explicitly or implicitly assigned to an existing database. If you create a table space and do not specify a database name, the table space is created in the default database, DSNDB04. In this case, Db2 implicitly creates a database or uses an existing implicitly created database for the table. All users who have the authority to create table spaces or tables in database DSNDB04 have authority to create tables and table spaces in an implicitly created database. If the table space is implicitly created, and you do not specify the IN clause in the CREATE TABLE statement, Db2 implicitly creates the database to which the table space is assigned.

A single database, for example, can contain all the data that is associated with one application or with a group of related applications. Collecting that data into one database allows you to start or stop access to all the data in one operation. You can also grant authorization for access to all the data as a single unit. Assuming that you are authorized to access data, you can access data that is stored in different databases.

Generally, it is best to minimize the number of table spaces in each database. For the recommended partition-by-range and partition-by-growth table space types, each table space contains only a single table. However, if you do use multi-table segmented (non-UTS) table spaces, which are deprecated, minimize the number of tables in each table space. Too many table spaces and tables in a database can cause decreases in performance and manageability issues. If you reduce the number of table spaces and tables in a database, you improve performance, minimize maintenance, increase concurrency, and decrease log volume.

Deprecated function: Non-UTS table spaces for base tables are deprecated and likely to be unsupported in the future.

The following figure shows how the main Db2 data structures fit together. Two databases, A and B, are represented as squares. Each database contains a table space and index space. Each index space contains one index. In database B, the table space is partitioned and contains table B1, partitions 1 - 4. The index space contains one partitioning index, parts 1 - 4.

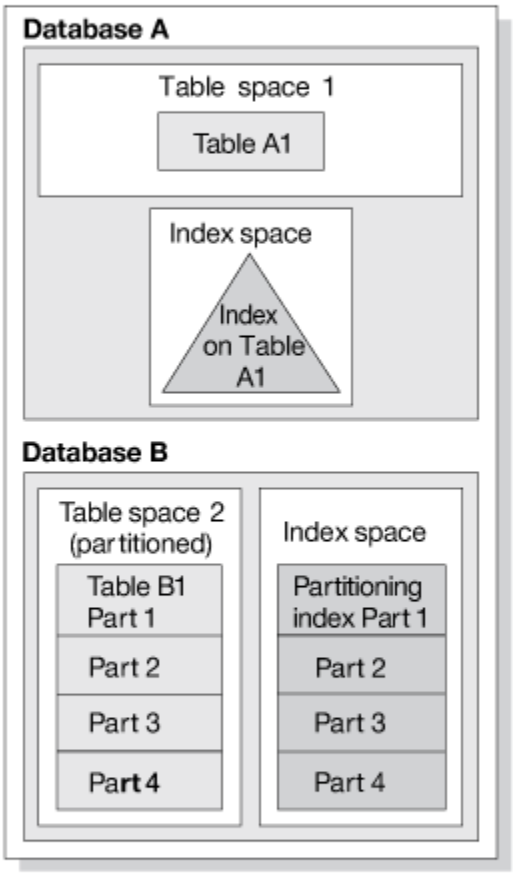


Figure 10. Data structures in a Db2 database

When you migrate Db2 11, Db2 adopts the default database and default storage group that you used in the previous version. You have the same authority for the Db2 11 as you did in the previous version.

Reasons to define a database

In Db2 for z/OS, a database is a logical collection of table spaces and index spaces. Consider the following factors when deciding whether to define a new database for a new set of objects:

- You can start and stop an entire database as a unit; you can display the statuses of all its objects by using a single command that names only the database. Therefore, place a set of tables that are used together into the same database. (The same database holds all indexes on those tables.)
- Some operations lock an entire database. For example, some phases of the LOAD utility prevent some SQL statements (CREATE, ALTER, and DROP) from using the same database concurrently. Therefore, placing many unrelated tables in a single database is often inconvenient.

  When one user is executing a CREATE, ALTER, or DROP statement for a table, no other user can access the database that contains that table. QMF users, especially, might do a great deal of data definition; the QMF operations SAVE DATA and ERASE *data-object* are accomplished by creating and dropping Db2 tables. For maximum concurrency, create a separate database for each QMF user.
- The internal database descriptors (DBDs) might become inconveniently large. DBDs grow as new objects are defined, but they do not immediately shrink when objects are dropped; the DBD space for a dropped object is not reclaimed until the MODIFY RECOVERY utility is used to delete records of obsolete copies from SYSIBM.SYSCOPY. DBDs occupy storage and are the objects of occasional input and output operations. Therefore, limiting the size of DBDs is another reason to define new databases.

Creation of databases
When you define a Db2 database, you name an eventual collection of tables, associated indexes, and the table spaces in which they are to reside.

Db2 table spaces

A Db2 *table space* is a set of volumes on disks that hold the data sets in which tables are actually stored. Every table is stored in table space.

A table space consists of a number of VSAM linear data sets. Table spaces are divided into equal-sized units, called *pages*. Each page is read from disk to the assigned buffer pool, or written from the buffer pool to disk, in a single operation. The page size is controlled by the buffer pool that you assign to the table space (4 KB, 8 KB, 16 KB, or 32 KB in size) for the data. The default page size is 4 KB.

Data in most table spaces can be compressed, which can allow you to store more data on each data page.

You can let Db2 create and manage the table space for you by issuing a CREATE TABLE statement that does not specify an existing table space. Db2 creates a partition-by-growth or partition-by-range table space. For more information, see “Creation of table spaces” on page 96.

You can also explicitly define table spaces by issuing CREATE TABLESPACE statements. You specify the database that the table space belongs to, and the storage group that it uses. When you create a table space, certain options that you specify control the type of table space that is created. It is best to create partition-by-growth or partition-by-range universal table spaces in most cases.

Deprecated function: Non-UTS table spaces for base tables are deprecated and likely to be unsupported in the future.

For descriptions of the various table space types, see Table space types and characteristics in Db2 for z/OS (Db2 Administration Guide).

Segmented non-UTS table spaces, which are deprecated, can contain more than one table. However, with the recommended partition-by-growth and partition-by-range table spaces, each table space contains only a single table. For smaller tables, multiple-table segmented table spaces can sometimes help to reduce the number of data sets that need to be managed for backup and recovery, and the number of data sets that the database system needs to open and close during Db2 operations.

Related tasks

Creating table spaces explicitly (Db2 Administration Guide)
Choosing data page sizes (Db2 Performance)

Related reference

ALTER TABLESPACE (Db2 SQL)

CREATE TABLESPACE (Db2 SQL)

Related information

Implementing Db2 table spaces (Db2 Administration Guide)

Conversion from index-controlled partitioning to Universal Table Space (UTS)

Db2 index spaces

An *index space* is a Db2 storage structure that contains a single index.

When you create an index by using the CREATE INDEX statement, an index space is automatically defined in the same database as the table. You can define a unique name for the index space, or Db2 can derive a unique name for you. Under certain circumstances, Db2 implicitly creates index spaces.

Rules for primary and secondary space allocation

You can specify the primary and secondary space allocation or let Db2 choose them. Having Db2 choose the values, especially the secondary space quantity, increases the possibility of reaching the maximum data set size before running out of extents.

In the following rules that describe how allocation works, these terms are used:

PRIQTY, SECQTY
The keywords for CREATE TABLESPACE, ALTER TABLESPACE, CREATE INDEX, and ALTER INDEX.

specified-priqty
The user-specified value for PRIQTY.

specified-secqty
The user-specified value for SECQTY.

actual-priqty
The actual primary space allocation, in kilobytes.

actual-priqty-cylinders
The actual primary space allocation, in cylinders.

actual-secqty
The actual secondary space allocation, in kilobytes.

actual-secqty-cylinders
The actual secondary space allocation, in cylinders.

calculated-extent-cylinders
A value that is calculated by Db2 using a *sliding scale*. A sliding scale means that the first secondary extent allocations are smaller than later secondary allocations. For example, Figure 11 on page 52 shows the sliding scale of secondary extent allocations that Db2 uses for a 64-GB data set. The size of each secondary extent is larger for each secondary extent that is allocated up to the 127th extent. For the 127th secondary extent and any subsequent extents, the secondary size allocation is 559 cylinders.

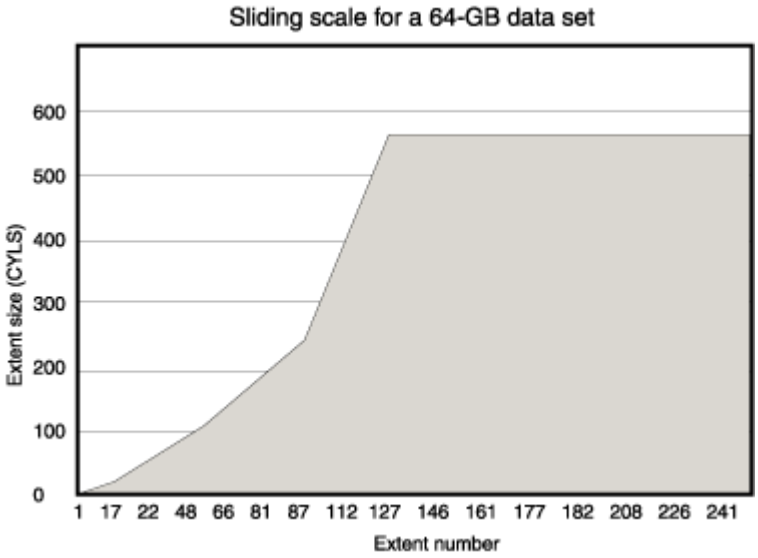


Figure 11. Sliding scale allocation of secondary extents for a 64 GB data set

Rules

Rule 1: (for primary space allocation)

If PRIQTY is specified and *specified-priqty* is not equal to -1, *actual-priqty* is at least *specified-priqty* KB.

If PRIQTY is not specified or *specified-priqty* is equal to -1, *actual-priqty* is determined as follows:

- For a table space, if the TSQTY subsystem parameter value is specified and is greater than 0, *actual-priqty* is at least the value of TSQTY.

  If the TSQTY subsystem parameter is not specified or is 0, *actual-priqty* is one cylinder for a non-LOB table space. *actual-priqty* is 10 cylinders for a LOB table space.
- For an index, if the IXQTY subsystem parameter value is specified and is greater than 0, *actual-priqty* is at least the value of IXQTY.

  If the IXQTY subsystem parameter is not specified or is 0, *actual-priqty* is one cylinder.

Rule 2: (for secondary space allocation)

If SECQTY is not specified, the following formulas determine *actual-secqty*:

- If the maximum size of a data set in the table space or index is less than 32 GB, the formula is:

```
actual-secqty-cylinders=
MAX(0.1*actual-priqty-cylinders, MIN(calculated-extent-cylinders, 127))
```

- If the maximum size of a data set in the table space or index is 32 GB or greater, the formula is:

```
actual-secqty-cylinders=
MAX(0.1*actual-priqty-cylinders, MIN(calculated-extent-cylinders, 559))
```

Rule 3 (for secondary space allocation)

If SECQTY is 0, *actual-secqty* is 0.

Rule 4 (for secondary space allocation)

This is the only rule that depends on the value of subsystem parameter MGEXTSZ (field OPTIMIZE EXTENT on installation panel DSNTIP7). If MGEXTSZ is YES:

- If SECQTY is specified and *specified-secqty* is not equal to -1 or 0, the following formulas determine *actual-secqty*. If the maximum size of a data set in the table space or index is less 32 GB, the formula is:

```
actual-secqty-cylinders=
MAX(MIN(calculated-extent-cylinders, 127),specified-secqty-cylinders)
```

  If the maximum size of a data set in the table space or index is 32 GB or greater, the formula is:

```
actual-secqty-cylinders=
MAX(MIN(calculated-extent-cylinders, 559),specified-secqty-cylinders)
```

If MGEXTSZ is NO:

- For a table space, if SECQTY is *n*, the secondary space allocation is at least *n* kilobytes, with the following exception: If SECQTY is greater than 4194304, *n* is 4194304 kilobytes.
- For an index, if SECQTY is *n*, the secondary space allocation is at least *n* kilobytes, where *n* is:

  12
  If SECQTY and PRIQTY are omitted

  4194304
  If *integer* is greater than 4194304

  integer
  If *integer* is not greater than 4194304

Rule 5 (for secondary space allocation):
When a table space requires a new piece, the primary allocation quantity of the new piece is determined as follows: If the value of subsystem parameter MGEXTSZ is NO, the primary quantity from rule 1 is used. Otherwise, the maximum of the following values is used:

- – The quantity that is calculated through sliding scale methodology
  – The primary quantity from rule 1
  – The specified SECQTY value

Related reference
CREATE TABLESPACE (Db2 SQL)
ALTER TABLESPACE (Db2 SQL)
CREATE INDEX (Db2 SQL)
ALTER INDEX (Db2 SQL)

Db2 hash spaces (deprecated)

A *hash space* is a defined disk space that organizes table data for hash access. Hash-organized table spaces are deprecated and likely to be unsupported in the future.

Deprecated function:

Hash-organized tables are deprecated. Beginning in Db2 12, packages that are bound with APPLCOMPAT(V12R1M504) or higher cannot create hash-organized tables or alter existing tables to use hash-organization. Existing hash-organized tables remain supported, but they are likely to be unsupported in the future.

When you organize a table for hash access, Db2 requires a defined amount of disk space to contain table data. You can specify the amount of disk space to allocate to the hash space when you create a table or alter an existing table. The hash space on a table must be large enough to contain new rows that are added to the table. If a hash space is full, new rows are relocated to the overflow index, which reduces the performance of hash access on that table. Hash spaces can contain only a single table in a partition-by-range or partition-by-growth table space.

Related concepts
Database design with hash access

You can organize tables for hash access to individual data rows. Hash-organized table spaces are deprecated and likely to be unsupported in the future.

Hash access paths
Db2 can use hash access paths to directly access a single row in a table and avoid scanning the table or index.

Related tasks

Managing space and page size for hash-organized tables (Db2 Performance)
Creating tables that use hash organization (deprecated) (Db2 Administration Guide)
Altering tables for hash access (deprecated) (Db2 Administration Guide)
Organizing tables for hash access to individual rows (deprecated) (Db2 Performance)
Monitoring hash access (deprecated) (Db2 Performance)

Db2 storage groups

Db2 *storage groups* are a set of volumes on disks that hold the data sets in which tables and indexes are stored.

The description of a storage group names the group and identifies its volumes and the VSAM (Virtual Storage Access Method) catalog that records the data sets. The default storage group, SYSDEFLT, is created when you install Db2.

Within the storage group, Db2 does the following actions:

- Allocates storage for table spaces and indexes
- Defines the necessary VSAM data sets
- Extends and deletes VSAM data sets
- Alters VSAM data sets

All volumes of a given storage group must have the same device type. However, parts of a single database can be stored in different storage groups.

Db2 can manage the auxiliary storage requirements of a database by using Db2 storage groups. Data sets in these Db2 storage groups are called "Db2-managed data sets."

These Db2 storage groups are not the same as storage groups that are defined by the DFSMS storage management subsystem (DFSMSsms).

You have several options for managing Db2 data sets:

- Let Db2 manage the data sets. This option means less work for Db2 database administrators.

  After you define a Db2 storage group, Db2 stores information about it in the Db2 catalog. (This catalog is not the same as the integrated catalog facility catalog that describes Db2 VSAM data sets). The catalog table SYSIBM.SYSSTOGROUP has a row for each storage group, and SYSIBM.SYSVOLUMES has a row for each volume. With the proper authorization, you can retrieve the catalog information about Db2 storage groups by using SQL statements.

  When you create table spaces and indexes, you name the storage group from which space is to be allocated. You can also assign an entire database to a storage group. Try to assign frequently accessed objects (indexes, for example) to fast devices, and assign seldom-used tables to slower devices. This approach to choosing storage groups improves performance.

  If you are authorized and do not take specific steps to manage your own storage, you can still define tables, indexes, table spaces, and databases. A default storage group, SYSDEFLT, is defined when Db2 is installed. Db2 uses SYSDEFLT to allocate the necessary auxiliary storage. Information about SYSDEFLT, as with any other storage group, is kept in the catalog tables SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES.

  For both user-managed and Db2-managed data sets, you need at least one integrated catalog facility (ICF) catalog; this catalog can be either a user catalog or a master catalog. These catalogs are created

with the ICF. You must identify the catalog of the ICF when you create a storage group or when you create a table space that does not use storage groups.

- Let SMS manage some or all the data sets, either when you use Db2 storage groups or when you use data sets that you have defined yourself. This option offers a reduced workload for Db2 database administrators and storage administrators. You can specify SMS classes when you create or alter a storage group.
- Define and manage your own data sets using VSAM Access Method Services. This option gives you the most control over the physical storage of tables and indexes.

Recommendation: Use Db2 storage groups and whenever you can, either specifically or by default. Also use SMS managed Db2 storage groups whenever you can.

Related tasks

Choosing data page sizes for LOB data (Db2 Performance)

Accessing Db2 data

You can retrieve data by using the SQL statement SELECT to specify a result table that can be derived from one or more tables.

In this information, examples of SQL statements illustrate how to code and use each clause of the SELECT statement to query a table. Examples of more advanced queries explain how to fine-tune your queries by using functions and expressions and how to query multiple tables with more complex statements that include unions, joins, and subqueries. The best way to learn SQL is to develop SQL statements like these examples and then execute them dynamically using a tool such as QMF for Workstation.

The data that is retrieved through SQL is always in the form of a table. Like the tables from which you retrieve the data, a result table has rows and columns. A program fetches this data one or more rows at a time.

GUPI

Example: Consider this SELECT statement:

```
SELECT LASTNAME, FIRSTNME
  FROM EMP
  WHERE DEPT = 'D11'
  ORDER BY LASTNAME;
```

This SELECT statement returns the following result table:

```
LASTNAME     FIRSTNME
========     ========
BROWN        DAVID
LUTZ         JENNIFER
STERN        IRVING
```

GUPI

Many of the examples in this information are based on the sample tables, which represent sample information about a computer company.

Related concepts

Db2 for z/OS and SQL concepts

Many structures and processes are associated with a relational database. The structures are the key components of a Db2 database system, and the processes are the interactions that occur when applications access the database system.

Related tasks

Accessing data from application programs (Db2 Application programming and SQL)

Related reference

Statements (Db2 SQL)

Ways to select data from columns

Several techniques are available for selecting columns from a database for your result tables.

There are several ways to select data from the columns in your table, but you must follow good practices for SELECT statements to guarantee good performance. When you write a SELECT statement, you must select only the rows and columns that your program needs, which reduces your CPU load and memory usage.

Selection of some columns

Select the columns that you want by specifying the name of each column. All columns appear in the order that you specify, not in their order in the table.

GUPI

Example: Notice that the DEPT table contains the DEPTNO column before the MGRNO column. Consider the following query:

```
SELECT MGRNO, DEPTNO
    FROM DSN8B10.DEPT
    WHERE ADMRDEPT = 'A00';
```

The result table looks like the following example:

```
MGRNO   DEPTNO
======  ======
000010  A00
000020  B01
000030  C01
------  D01
000050  E01
```

This SELECT statement retrieves data that is contained in the two specified columns of each row in the DEPT table. You can select data from up to 750 columns with a single SELECT statement.

GUPI

Selection of all columns

You do not need to know the column names to select Db2 data. Use an asterisk (*) in the SELECT clause to retrieve all columns from each selected row of the specified table. Db2 selects the columns in the order that the columns are declared in that table. Hidden columns, such as ROWID columns and XML document ID columns, are not included in the result of the SELECT * statement.

GUPI

Example: Consider this query:

```
SELECT *
    FROM DSN8A10.DEPT
    WHERE ADMRDEPT = 'A00';
```

The result table looks like the following example:

```
DEPTNO  DEPTNAME                             MGRNO   ADMRDEPT  LOCATION
======  ========                             =====   ========  ========
A00     SPIFFY COMPUTER SERVICE DIV.         000010  A00
B01     PLANNING                             000020  A00
C01     INFORMATION CENTER                   000030  A00
D01     DEVELOPMENT CENTER                   ------  A00
E01     SUPPORT SERVICES                     000050  A00
```

This SELECT statement retrieves data from each column of each retrieved row of the DEPT table. Because the example does not specify a WHERE clause, the statement retrieves data from all rows.

In this example, the fifth row contains a null value because no manager is identified for this department. Null values are displayed as dashes.

GUPI

The SELECT * statement is most appropriate when used with dynamic SQL and view definitions. Avoid using SELECT * in static SQL. You write static SQL applications when you know the number of columns that your application returns. That number can change outside your application. If a change occurs to the table, you need to update the application to reflect the changed number of columns in the table.

Use the SELECT * statement only when it is necessary to retrieve all the columns in each retrieved row of your table. Selecting specific columns give your query a higher filter that can retrieve your results more efficiently.

Elimination of duplicate rows

The DISTINCT keyword removes redundant duplicate rows from your result table so that each row contains unique data. The following query uses the DISTINCT keyword to list the department numbers of the different administrative departments:

GUPI

```
SELECT DISTINCT ADMRDEPT
  FROM DSN8B10.DEPT;
```

The result table looks like the following example:

```
ADMRDEPT
========
A00
D11
E01
```

GUPI

You can use more than one DISTINCT keyword in a single query.

Selection of derived columns and naming the resulting columns

You can select columns that are derived from a constant, an expression, or a function. With the AS clause, you can name resulting columns. This keyword is useful for a column that is derived from an expression or a function.

GUPI

Example: In the following query, the expression SALARY+COMM is named TOTAL_SAL:

```
SELECT EMPNO, (SALARY + COMM) AS TOTAL_SAL
    FROM DSN8B10.EMP;
```

The result table looks like the following example:

```
EMPNO      TOTAL_SAL
======     =========
000290      16567.00
000310      17172.00
200310      17172.00
000260      18630.00
000300      19170.00
000210      19732.00
⋮
```

This query selects data from all rows in the EMP table, calculates the result of the expression, and returns the columns in the order that the SELECT statement indicates. In the result table, any derived columns, such as (SALARY + COMM) in this example, do not have names. You can use the AS clause to give names to unnamed columns.

To order the rows in the result table by the values in a derived column, specify a name for the column by using the AS clause and use that name in the ORDER BY clause.

GUPI

Related concepts

Ways to filter the number of returned rows
A variety of different comparison operators in the predicate of a WHERE clause let you filter the number of returned rows.

Retrieving and excluding rows with null values
A *null value* indicates the absence of a column value in a row. A null value is not the same as zero or all blanks. You can retrieve or exclude rows that contain a null value in a specific row.

Ways to order rows
You can use the ORDER BY clause to retrieve rows in a specific order.

Related tasks

Retrieving data by using the SELECT statement (Db2 Application programming and SQL)

Related reference

select-statement (Db2 SQL)

How a SELECT statement works

SQL statements, including SELECT, are made up a series of clauses that are defined by SQL as being executed in a logical order. SELECT statements allow users to definite and organize information that is retrieved from a specified table.

GUPI

The following clause list shows the logical order of clauses in a statement:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

GUPI

In addition:

- *Subselects* are processed from the innermost to the outermost subselect. A subselect in a WHERE clause or a HAVING clause of another SQL statement is called a *subquery*.
- The ORDER BY clause can be included in a subselect, a fullselect, or in a SELECT statement.
- If you use an AS clause to define a name in the outermost SELECT clause, only the ORDER BY clause can refer to that name. If you use an AS clause in a subselect, you can refer to the name that it defines outside the subselect.

GUPI

Example 1: Consider this SELECT statement, which is not valid:

```
SELECT EMPNO, (SALARY + COMM) AS TOTAL_SAL
  FROM EMP
  WHERE TOTAL_SAL> 50000;
```

The WHERE clause is not valid because Db2 does not process the AS TOTAL_SAL portion of the statement until after the WHERE clause is processed. Therefore, Db2 does not recognize the name TOTAL_SAL that the AS clause defines.

Example 2: The following SELECT statement, however, is valid because the ORDER BY clause refers to the column name TOTAL_SAL that the AS clause defines:

```
SELECT EMPNO, (SALARY + COMM) AS TOTAL_SAL
  FROM EMP
  ORDER BY TOTAL_SAL;
```

GUPI

Related tasks

Coding SQL statements to avoid unnecessary processing (Db2 Performance)

Related reference

select-statement (Db2 SQL)

SQL functions and expressions

You can use functions and expressions to control the appearance and values of rows and columns in your result tables. Db2 offers many built-in functions, including aggregate functions and scalar functions.

A *built-in function* is a function that is supplied with Db2 for z/OS.

Concatenation of strings

You can concatenate strings by using the CONCAT operator or the CONCAT built-in function.

When the operands of two strings are concatenated, the result of the expression is a string. The operands of concatenation must be compatible strings.

GUPI

Example: Consider this query:

```
SELECT LASTNAME CONCAT ',' CONCAT FIRSTNME
  FROM EMP;
```

This SELECT statement concatenates the last name, a comma, and the first name of each result row. The result table looks like this:

```
================
HAAS,CHRISTINE
THOMPSON,MICHAEL
KWAN,SALLY
STERN,IRVING
⋮
```

Alternative syntax for the SELECT statement shown above is as follows:

```
SELECT LASTNAME CONCAT(CONCAT(LASTNAME,','),FIRSTNME)
FROM EMP;
```

In this case, the SELECT statement concatenates the last name and then concatenates that result to the first name.

GUPI

Related concepts

Expressions with the concatenation operator (Db2 SQL)

Related reference

CONCAT (Db2 SQL)

select-statement (Db2 SQL)

Calculation of values in one or more columns

You can perform calculations on numeric or datetime data.

The numeric data types are binary integer, floating-point, and decimal. The datetime data types are date, time, and timestamp.

You can retrieve calculated values, just as you display column values, for selected rows.

GUPI

Example: Consider this query:

```
SELECT EMPNO,
  SALARY / 12 AS MONTHLY_SAL,
  SALARY / 52 AS WEEKLY_SAL
  FROM DSN8B10.EMP
  WHERE WORKDEPT = 'A00';
```

The result table looks like the following example:

```
EMPNO       MONTHLY_SAL         WEEKLY_SAL
=====            ===========                ==========
000010      4395.83333333       1014.42307692
000110      3875.00000000       894.23076923
000120      2437.50000000       562.50000000
200010      3875.00000000       894.23076923
200120      2437.50000000       562.50000000
```

The result table displays the monthly and weekly salaries of employees in department A00. If you prefer results with only two digits to the right of the decimal point, you can use the DECIMAL function.

Example: To retrieve the department number, employee number, salary, and commission for those employees whose combined salary and commission is greater than $45,000, write the query as follows:

```
SELECT WORKDEPT, EMPNO, SALARY, COMM
  FROM DSN8B10.EMP
  WHERE SALARY + COMM > 45000;
```

The result table looks like following example:

```
DEPT     EMPNO      SALARY        COMM
====     ======     ========      =======
A00      000010     52750.00      4220.00
A00      000110     46500.00      3720.00
A00      200010     46500.00      4220.00
```

GUPI

Related concepts

Scalar functions

Db2 offers many different scalar functions, including the CHAR, DECIMAL, and NULLIF scalar functions.

Calculation of aggregate values

You can use the SQL aggregate functions to calculate values that are based on entire columns of data. The calculated values are from only the rows that satisfy the WHERE clause and are therefore selected.

An *aggregate function* is an operation that derives its result by using values from one or more rows. An aggregate function is also known as a *column function*. The argument of an aggregate function is a set of values that are derived from an expression.

You can use the following aggregate functions:

SUM
Returns the total value.

MIN
Returns the minimum value.

AVG
Returns the average value.

MAX
Returns the maximum value.

COUNT
Returns the number of selected rows.

COUNT_BIG
Returns the number of rows or values in a set of rows or values. The result can be greater than the maximum value of an integer.

XMLAGG
Returns a concatenation of XML elements from a collection of XML elements.

GUPI

Example 1: This query calculates, for department A00, the sum of employee salaries, the minimum, average, and maximum salary, and the count of employees in the department:

```
SELECT SUM(SALARY) AS SUMSAL,
       MIN(SALARY) AS MINSAL,
       AVG(SALARY) AS AVGSAL,
       MAX(SALARY) AS MAXSAL,
       COUNT(*) AS CNTSAL
  FROM EMP
  WHERE DEPT = 'A00';
```

The result table looks like this:

```
   SUMSAL       MINSAL            AVGSAL       MAXSAL     CNTSAL
=========     ========     ==============     ========     ======
128500.00     29250.00     42833.33333333     52750.00          3
```

You can use (*) in the COUNT and COUNT_BIG functions. In this example, COUNT(*) returns the rows that Db2 processes based on the WHERE clause.

Example 2: This query counts the number of employees that are described in the EMP table:

```
SELECT COUNT(*)
  FROM EMP;
```

You can use DISTINCT with the SUM, AVG, COUNT, and COUNT_BIG functions. DISTINCT means that the selected function operates on only the unique values in a column.

Example 3: This query counts the different jobs in the EMP table:

```
SELECT COUNT(DISTINCT JOB)
  FROM EMP;
```

GUPI

Aggregate functions like COUNT ignore nulls in the values on which they operate. The preceding example counts distinct job values that are not null.

Note: Do not use DISTINCT with the MAX and MIN functions because using it does not affect the result of those functions.

You can use SUM and AVG only with numbers. You can use MIN, MAX, COUNT, and COUNT_BIG with any built-in data type.

Related reference
DECIMAL or DEC (Db2 SQL)

Scalar functions
Db2 offers many different scalar functions, including the CHAR, DECIMAL, and NULLIF scalar functions.

Like an aggregate function, a *scalar function* produces a single value. Unlike the argument of an aggregate function, an argument of a scalar function is a single value.

GUPI

Example: YEAR: This query, which uses the YEAR scalar function, returns the year in which each employee in a particular department was hired:

```
SELECT YEAR(HIREDATE) AS HIREYEAR
  FROM EMP
  WHERE DEPT = 'A00';
```

The result table looks like this:

```
HIREYEAR
========
    1975
    1990
    1985
```

The YEAR scalar function produces a single scalar value for each row of EMP that satisfies the search condition. In this example, three rows satisfy the search condition, so YEAR results in three scalar values.

Db2 offers many different scalar functions, including CHAR, DECIMAL, and NULLIF.

CHAR

The CHAR function returns a string representation of the input value.

Example: CHAR: The following SQL statement sets the host variable AVERAGE to the character string representation of the average employee salary:

```
SELECT CHAR(AVG(SALARY))
  INTO :AVERAGE
  FROM EMP;
```

DECIMAL

The DECIMAL function returns a decimal representation of the input value.

Example: DECIMAL: Assume that you want to change the decimal data type to return a value with a precision and scale that you prefer. The following example represents the average salary of employees as an eight-digit decimal number (the precision) with two of these digits to the right of the decimal point (the scale):

```
SELECT DECIMAL(AVG(SALARY),8,2)
  FROM EMP;
```

The result table looks like this:

```
==========
  32602.30
```

NULLIF

NULLIF returns a null value if the two arguments of the function are equal. If the arguments are not equal, NULLIF returns the value of the first argument.

Example: NULLIF: Suppose that you want to calculate the average earnings of all employees who are eligible to receive a commission. All eligible employees have a commission of greater than 0, and ineligible employees have a value of 0 for commission:

```
SELECT AVG(SALARY+NULLIF(COMM,0))
  AS "AVERAGE EARNINGS"
  FROM EMP;
```

The result table looks like this:

```
AVERAGE EARNINGS
================
   35248.8461538
```

GUPI

Specifying a simple expression for the sum of the salary and commission in the select list would include all employees in the calculation of the average. To avoid including those employees who do not earn a commission in the average, you can use the NULLIF function to return a null value instead. The result of

adding a null value for the commission to SALARY is itself a null value, and aggregate functions, like AVG, ignore null values. Therefore, this use of NULLIF inside AVG causes the query to exclude each row in which the employee is not eligible for a commission.

Related reference

Scalar functions (Db2 SQL)

Nested functions

Scalar and aggregate functions can be nested in several ways.

You can nest functions in the following ways:

- Scalar functions within scalar functions GUPI

  Example: Suppose that you want to know the month and day of hire for a particular employee in department D11. Suppose that you also want the result in USA format (mm/dd/yyyy). Use this query:

```
SELECT SUBSTR((CHAR(HIREDATE, USA)),1,5)
  FROM EMP
  WHERE LASTNAME = 'BROWN' AND DEPT = 'D11';
```

  The result table looks like this:

```
=====
03/03
```

  GUPI

- Scalar functions within aggregate functions

  In some cases, you might need to invoke a scalar function from within an aggregate function.

  GUPI

  Example: Suppose that you want to know the average number of years of employment for employees in department A00. Use this query:

```
SELECT AVG(DECIMAL(YEAR(CURRENT DATE - HIREDATE)))
  FROM EMP
  WHERE DEPT = 'A00';
```

  The result table looks like this:

```
=======
20.6666
```

  The actual form of the result, 20.6666, depends on how you define the host variable to which you assign the result.

  GUPI

- Aggregate functions within scalar functions GUPI

  Example: Suppose that you want to know the year in which the last employee was hired in department E21. Use this query:

```
SELECT YEAR(MAX(HIREDATE))
  FROM EMP
  WHERE DEPT = 'E21';
```

  The result table looks like this:

```
====
2002
```

  GUPI

Related concepts

Date, time, and timestamp data types
Although storing dates and times as numeric values is possible, using datetime data types is recommended. The datetime data types are DATE, TIME, and TIMESTAMP.

Aggregate functions (Db2 SQL)

Related reference

Scalar functions (Db2 SQL)

User-defined functions

User-defined functions are small programs that you can write to perform an operation. You can use a user-defined function wherever you can use a built-in function.

The CREATE FUNCTION statement is used to explicitly create a user-defined function.

Example: Assume that you define a distinct type called US_DOLLAR. You might want to allow instances of US_DOLLAR to be added. You can create a user-defined function that uses a built-in addition operation and takes instances of US_DOLLAR as input. This function, called a *sourced* function, requires no application coding. Alternatively, you might create a more complex user-defined function that can take a US_DOLLAR instance as input and then convert from U.S. dollars to another currency.

You name the function and specify its semantics so that the function satisfies your specific programming needs. You can use a user-defined function wherever you can use a built-in function.

Related concepts

Creation of user-defined functions
You can create your own functions in Db2 to simplify your queries.

User-defined functions (Db2 SQL)

Types of functions (Db2 SQL)

Sample user-defined functions (Db2 SQL)

Related tasks

Creating a user-defined function (Db2 Application programming and SQL)

CASE expressions

You can use a CASE expression to execute SQL expressions in several different ways depending on the value of a search condition.

One use of a CASE expression is to replace the values in a result table with more meaningful values.

GUPI

Example: Suppose that you want to display the employee number, name, and education level of all field representatives in the EMP table. Education levels are stored in the EDL column as small integers, but you want to replace the values in this column with more descriptive phrases. Use the following query:

```
SELECT EMPNO, FIRSTNME, LASTNAME,
  CASE
    WHEN EDL<=12 THEN 'HIGH SCHOOL OR LESS'
    WHEN EDL>12 AND EDL<=14 THEN 'JUNIOR COLLEGE'
    WHEN EDL>14 AND EDL<=17 THEN 'FOUR-YEAR COLLEGE'
    WHEN EDL>17 THEN 'GRADUATE SCHOOL'
    ELSE 'UNKNOWN'
  END
  AS EDUCATION
  FROM EMP
  WHERE JOB='FLD';
```

The result table looks like following example:

```
EMPNO      FIRSTNME     LASTNAME     EDUCATION
======     ========     ========     =================
000320     RAMLAL       MEHTA        FOUR-YEAR COLLEGE
000330     WING         LEE          JUNIOR COLLEGE
200340     ROY          ALONZO       FOUR-YEAR COLLEGE
```

The CASE expression replaces each small integer value of EDL with a description of the amount of each field representative's education. If the value of EDL is null, the CASE expression substitutes the word UNKNOWN.

Another use of a CASE expression is to prevent undesirable operations, such as division by zero, from being performed on column values.

Example: If you want to determine the ratio of employee commissions to their salaries, you can execute this query:

```
SELECT EMPNO, DEPT,
  COMM/SALARY AS "COMMISSION/SALARY",
FROM EMP;
```

This SELECT statement has a problem, however. If an employee has not earned any salary, a division-by-zero error occurs. By modifying the following SELECT statement with a CASE expression, you can avoid division by zero:

```
SELECT EMPNO, DEPT,
  (CASE WHEN SALARY=0 THEN NULL
    ELSE COMM/SALARY
   END) AS "COMMISSION/SALARY"
  FROM EMP;
```

The CASE expression determines the ratio of commission to salary only if the salary is not zero. Otherwise, Db2 sets the ratio to a null value.

GUPI

Related reference

CASE expressions (Db2 SQL)

Ways to filter the number of returned rows

A variety of different comparison operators in the predicate of a WHERE clause let you filter the number of returned rows.

You can use a WHERE clause to select the rows that are of interest to you. For example, suppose you want to select only the rows that represent the employees who earn a salary greater than $40,000. A WHERE clause specifies a *search condition*. A search condition is the criteria that Db2 uses to select rows. For any given row, the result of a search condition is true, false, or unknown. If the search condition evaluates to true, the row qualifies for additional processing. In other words, that row can become a row of the result table that the query returns. If the condition evaluates to false or unknown, the row does not qualify for additional processing.

A search condition consists of one or more *predicates* that are combined through the use of the logical operators AND, OR, and NOT. An individual predicate specifies a test that you want Db2 to apply to each row, for example, `SALARY> 40000`. When Db2 evaluates a predicate for a row, it evaluates to true, false, or unknown. Results are unknown only if a value (called an operand) of the predicate is null. If a particular employee's salary is not known (and is set to null), the result of the predicate `SALARY> 40000` is unknown.

You can use a variety of different comparison operators in the predicate of a WHERE clause, as shown in the following table.

Table 9. Comparison operators used in conditions

Type of comparison	Specified with...	Example of predicate with comparison
Equal to null	IS NULL	COMM IS NULL
Equal to	=	DEPTNO = 'X01'
Not equal to	<>	DEPTNO <> 'X01'

Table 9. Comparison operators used in conditions (continued)

Type of comparison	Specified with...	Example of predicate with comparison
Less than	<	AVG(SALARY) < 30000
Less than or equal to	<=	SALARY <= 50000
Greater than	>	SALARY> 25000
Greater than or equal to	>=	SALARY>= 50000
Similar to another value	LIKE	NAME LIKE ' or STATUS LIKE 'N_'
At least one of two predicates	OR	HIREDATE < '2000-01-01' OR SALARY < 40000
Both of two predicates	AND	HIREDATE < '2000-01-01' AND SALARY < 40000
Between two values	BETWEEN	SALARY BETWEEN 20000 AND 40000
Equals a value in a set	IN (X, Y, Z)	DEPTNO IN ('B01', 'C01', 'D11')
Compares a value to another value	DISTINCT	value 1 IS DISTINCT from value 2

Note: Another predicate, EXISTS, tests for the existence of certain rows. The result of the predicate is true if the result table that is returned by the subselect contains at least one row. Otherwise, the result is false.

The XMLEXISTS predicate can be used to restrict the set of rows that a query returns, based on the values in XML columns. The XMLEXISTS predicate specifies an XPath expression. If the XPath expression returns an empty sequence, the value of the XMLEXISTS predicate is false. Otherwise, XMLEXISTS returns true. Rows that correspond to an XMLEXISTS value of true are returned.

You can also search for rows that do not satisfy one of the predicates by using the NOT keyword before the specified predicate.

Related concepts
Predicates (Db2 SQL)
Related tasks
Using predicates efficiently (Db2 Performance)
Related reference
Summary of predicate processing (Db2 Performance)
where-clause (Db2 SQL)

Retrieving and excluding rows with null values
A *null value* indicates the absence of a column value in a row. A null value is not the same as zero or all blanks. You can retrieve or exclude rows that contain a null value in a specific row.
GUPI

Example 1: You can use a WHERE clause to retrieve rows that contain a null value in a specific column. Specify:

```
WHERE column-name IS NULL
```

Example 2: You can also use a predicate to exclude null values. Specify:

```
WHERE column-name IS NOT NULL
```

GUPI

You cannot use the equal sign to retrieve rows that contain a null value. (WHERE *column-name* = NULL is not allowed.)

Equalities and inequalities
You can use an equal sign (=), various inequality symbols, and the NOT keyword to specify search conditions in the WHERE clause.

Related tasks

Using predicates efficiently (Db2 Performance)

Related reference

Basic predicate (Db2 SQL)

where-clause (Db2 SQL)

How to test for equality

You can use an equal sign (=) to select rows for which a specified column contains a specified value.

GUPI

To select only the rows where the department number is A00, use WHERE DEPT = 'A00' in your SELECT statement:

```
SELECT FIRSTNME, LASTNAME
  FROM EMP
  WHERE DEPT = 'A00';
```

This query retrieves the first and last name of each employee in department A00.

GUPI

How to test for inequalities

You can use inequality operators to specify search conditions in your SELECT statements.

GUPI

You can use the following inequalities to specify search conditions:

```
<>  <  <=  >  >=
```

Example: To select all employees that were hired before January 1, 2001, you can use this query:

```
SELECT HIREDATE, FIRSTNME, LASTNAME
  FROM EMP
  WHERE HIREDATE < '2001-01-01';
```

This SELECT statement retrieves the hire date and name for each employee that was hired before 2001.

GUPI

How to test for equality or inequality in a set of columns

You can use the equal operator or the not equal operator to test whether a set of columns is equal or not equal to a set of values.

Example 1: To select the rows in which the department number is A00 and the education level is 14, you can use this query:

```
SELECT FIRSTNME, LASTNAME
  FROM EMP
  WHERE (DEPT, EDL) = ('A00', 14);
```

Example 2: To select the rows in which the department number is not A00, or the education level is not 14, you can use this query:

```
SELECT FIRSTNME, LASTNAME
  FROM EMP
  WHERE (DEPT, EDL) <> ('A00', 14);
```

How to test for a false condition

You can use the NOT keyword to test for a false condition.

You can use the NOT keyword to select all rows for which the predicate is false (but not rows for which the predicate is unknown). The NOT keyword must precede the predicate.

GUPI

Example: To select all managers whose compensation is not greater than $40,000, use:

```
SELECT DEPT, EMPNO
  FROM EMP
  WHERE NOT (SALARY + COMM)> 40000 AND JOB = 'MGR'
  ORDER BY DEPT;
```

The following table contrasts WHERE clauses that use a NOT keyword with comparison operators and WHERE clauses that use only comparison operators. The WHERE clauses are equivalent.

Table 10. Equivalent WHERE clauses

Using NOT	Equivalent clause without NOT
WHERE NOT DEPTNO = 'A00'	WHERE DEPTNO <> 'A00'
WHERE NOT DEPTNO < 'A00'	WHERE DEPTNO>= 'A00'
WHERE NOT DEPTNO> 'A00'	WHERE DEPTNO <= 'A00'
WHERE NOT DEPTNO <> 'A00'	WHERE DEPTNO = 'A00'
WHERE NOT DEPTNO <= 'A00'	WHERE DEPTNO> 'A00'
WHERE NOT DEPTNO>= 'A00'	WHERE DEPTNO < 'A00'

You cannot use the NOT keyword directly preceding equality and inequality comparison operators.

Example: The following WHERE clause results in an error:

```
Wrong:
WHERE DEPT NOT = 'A00'
```

Example: The following two clauses are equivalent:

```
Correct:
WHERE MGRNO NOT IN ('000010', '000020')
WHERE NOT MGRNO IN ('000010', '000020')
```

GUPI

Similarities of character data

You can use the LIKE predicate to specify a character string that is similar to the column value of rows that you want to select.

A LIKE pattern must match the character string in its entirety.

- Use a percent sign (%) to indicate any string of zero or more characters.
- Use an underscore (_) to indicate any single character.

You can also use NOT LIKE to specify a character string that is not similar to the column value of rows that you want to select.

How to select values similar to a string of unknown characters

The percent sign (%) means "any string or no string."

GUPI

Example: The following query selects data from each row for employees with the initials D B:

```
SELECT FIRSTNME, LASTNAME, DEPT
  FROM EMP
  WHERE FIRSTNME LIKE 'D%' AND LASTNAME LIKE 'B
```

Example: The following query selects data from each row of the department table, where the department name contains "CENTER" anywhere in its name:

```
SELECT DEPTNO, DEPTNAME
  FROM DEPT
  WHERE DEPTNAME LIKE '
```

Example: Assume that the DEPTNO column is a three-character column of fixed length. You can use the following search condition to return rows with department numbers that begin with E and end with 1:

```
…WHERE DEPTNO LIKE 'E%1';
```

In this example, if E1 is a department number, its third character is a blank and does not match the search condition. If you define the DEPTNO column as a three-character column of varying length instead of fixed length, department E1 would match the search condition. Varying-length columns can have any number of characters, up to and including the maximum number that was specified when the column was created.

Example: The following query selects data from each row of the department table, where the department number starts with an E and contains a 1:

```
SELECT DEPTNO, DEPTNAME
  FROM DEPT
  WHERE DEPTNO LIKE 'E
```

GUPI

How to select a value similar to a single unknown character

The underscore (_) means "any single character."

GUPI

Example: Consider the following query:

```
SELECT DEPTNO, DEPTNAME
   FROM DEPT
   WHERE DEPTNO LIKE 'E_1';
```

In this example, 'E_1' means E, followed by any character, followed by 1. (Be careful: '_' is an underscore character, not a hyphen.) 'E_1' selects only three-character department numbers that begin with E and end with 1; it does not select the department number 'E1'.

GUPI

Related concepts

String data types
Db2 supports several types of string data: character strings, graphic strings, and binary strings.

Multiple conditions

You can use the AND and OR operators to combine predicates and search for data, based on multiple conditions.

Use the AND operator to specify that a search must satisfy both of the conditions. Use the OR operator to specify that the search must satisfy at least one of the conditions.

GUPI

Example: This query retrieves the employee number, hire date, and salary for each employee who was hired before 1998 and earns a salary of less than $35,000 per year:

```
SELECT EMPNO, HIREDATE, SALARY
  FROM EMP
  WHERE HIREDATE < '1998-01-01' AND SALARY < 35000;
```

Example: This query retrieves the employee number, hire date, and salary for each employee who either was hired before 1998, or earns a salary less than $35,000 per year or both

Note: :

```
SELECT EMPNO, HIREDATE, SALARY
  FROM EMP
  WHERE HIREDATE < '1998-01-01' OR SALARY < 35000;
```

GUPI

How to use parentheses with AND and OR

If you use more than two conditions with the AND or OR operators, you can use parentheses to specify the order in which you want Db2 to evaluate the search conditions. If you move the parentheses, the meaning of the WHERE clause can change significantly.

GUPI

Example: This query retrieves the row of each employee that satisfies at least one of the following conditions:

- The employee's hire date is before 1998 and salary is less than $40,000.
- The employee's education level is less than 18.

```
SELECT EMPNO
  FROM EMP
  WHERE (HIREDATE < '1998-01-01' AND SALARY < 40000) OR (EDL < 18);
```

Example: This query retrieves the row of each employee that satisfies both of the following conditions:

- The employee's hire date is before 1998.
- The employee's salary is less than $40,000 or the employee's education level is less than 18.

```
SELECT EMPNO
  FROM EMP
  WHERE HIREDATE < '1998-01-01' AND (SALARY < 40000 OR EDL < 18);
```

Example: This query retrieves the employee number of each employee that satisfies one of the following conditions:

- Hired before 1998 and salary is less than $40,000.
- Hired after January 1, 1998, and salary is greater than $40,000.

```
SELECT EMPNO
  FROM EMP
  WHERE (HIREDATE < '1998-01-01' AND SALARY < 40000)
  OR (HIREDATE> '1998-01-01' AND SALARY> 40000);
```

GUPI

How to use NOT with AND and OR

When you use NOT with AND and OR, the placement of the parentheses is important.

GUPI

Example: The following query retrieves the employee number, education level, and job title of each employee who satisfies both of the following conditions:

- The employee's salary is less than $50,000.
- The employee's education level is less than 18.

```
SELECT EMPNO, EDL, JOB
  FROM EMP
  WHERE NOT (SALARY>= 50000) AND (EDL < 18);
```

In this query, the NOT operator affects only the first search condition (SALARY>= 50000).

Example: The following query retrieves the employee number, education level, and job title of each employee who satisfies at least one of the following conditions:

- The employee's salary is less than or equal to $50,000.
- The employee's education level is less than or equal to 18.

```
SELECT EMPNO, EDL, JOB
  FROM EMP
  WHERE NOT (SALARY> 50000 AND EDL> 18);
```

To negate a set of predicates, enclose the entire set in parentheses and precede the set with the NOT keyword.

GUPI

Ranges of values

You can use BETWEEN to select rows in which a column has a value within two limits.

Specify the lower boundary of the BETWEEN predicate first, and then specify the upper boundary. The limits are *inclusive*.

GUPI

Example: Suppose that you specify the following WHERE clause in which the value of the *column-name* column is an integer:

```
WHERE column-name BETWEEN 6 AND 8
```

Db2 selects all rows whose *column-name* value is 6, 7, or 8. If you specify a range from a larger number to a smaller number (for example, BETWEEN 8 AND 6), the predicate never evaluates to true.

GUPI GUPI

Example: This query retrieves the department number and manager number of each department whose number is between C00 and D31:

```
SELECT DEPTNO, MGRNO
  FROM DEPT
  WHERE DEPTNO BETWEEN 'C00' AND 'D31';
```

GUPI

You can also use NOT BETWEEN to select rows in which a column has a value that is outside the two limits.

Values in a list

You can use the IN predicate to select each row that has a column value that is equal to one of several listed values.

In the values list after the IN predicate, the order of the items is not important and does not affect the ordering of the result. Enclose the entire list of values in parentheses, and separate items by commas; the blanks are optional.

GUPI

Example: The following query retrieves the department number and manager number for departments B01, C01, and D11:

```
SELECT DEPTNO, MGRNO
  FROM DEPT
  WHERE DEPTNO IN ('B01', 'C01', 'D11');
```

Using the IN predicate gives the same results as a much longer set of conditions that are separated by the OR keyword.

Example: You can alternatively code the WHERE clause in the SELECT statement in the previous example as:

```
WHERE DEPTNO = 'B01' OR DEPTNO = 'C01' OR DEPTNO = 'D11';
```

However, the IN predicate saves coding time and is easier to understand.

Example: The following query finds the projects that do not include employees in department C01 or E21:

```
SELECT PROJNO, PROJNAME, RESPEMP
  FROM PROJ
  WHERE DEPTNO NOT IN ('C01', 'E21');
```

GUPI

Related concepts

IN-list access (ACCESSTYPE='N' or 'IN') (Db2 Performance)

Related reference

IN predicate (Db2 SQL)

Ways to order rows

You can use the ORDER BY clause to retrieve rows in a specific order.

Using ORDER BY is the only way to guarantee that your rows are in the sequence in which you want them. This information demonstrates how to use the ORDER BY clause.

Related tasks

Formatting the result table (Db2 Application programming and SQL)

Coding SQL statements to avoid unnecessary processing (Db2 Performance)

Related reference

order-by-clause (Db2 SQL)

Sort key

You can specify the order of selected rows by using sort keys that you identify in the ORDER BY clause.

A sort key can be a column name, an integer that represents the number of a column in the result table, or an expression. You can identify more than one column.

You can list the rows in ascending or descending order. Null values are included last in an ascending sort and first in a descending sort.

Db2 sorts strings in the collating sequence that is associated with the encoding scheme of the table. Db2 sorts numbers algebraically and sorts datetime values chronologically.

Related tasks

Ordering the result table rows (Db2 Application programming and SQL)

Related reference

order-by-clause (Db2 SQL)

Ascending order

You can retrieve results in ascending order by specifying ASC in the ORDER BY clause of your SELECT statement.

GUPI

Example: The following query retrieves the employee numbers, last names, and hire dates of employees in department A00 in ascending order of hire dates:

```
SELECT EMPNO, LASTNAME, HIREDATE
  FROM EMP
  WHERE DEPT = 'A00'
  ORDER BY HIREDATE ASC;
```

The result table looks like this:

```
EMPNO      LASTNAME      HIREDATE
======     =========     ==========
000010     HAAS          1975-01-01
200010     HEMMINGER     1985-01-01
000120     CONNOR        1990-12-05
```

This SELECT statement shows the seniority of employees. ASC is the default sorting order.

GUPI

Related reference

order-by-clause (Db2 SQL)

Descending order

You can retrieve results in descending order by specifying DESC in the ORDER BY clause.

GUPI

Example: This query retrieves department numbers, last names, and employee numbers in descending order of department number:

```
SELECT DEPT, LASTNAME, EMPNO
  FROM EMP
  WHERE JOB = 'SLS'
  ORDER BY DEPT DESC;
```

The result table looks like this:

```
DEPT     LASTNAME     EMPNO
====     =========    ======
C01      NICHOLLS     000140
A00      HEMMINGER    200010
A00      CONNOR       000120
```

GUPI

Related reference

order-by-clause (Db2 SQL)

Sort keys with multiple columns

You can specify more than one column name in the ORDER BY clause to order rows by the values in more than one column.

When several rows have the same first ordering column value, those rows are in order of the second column that you identify in the ORDER BY clause, and then on the third ordering column, and so on.

GUPI

Example: Consider this query:

```
SELECT JOB, EDL, LASTNAME
  FROM EMP
```

```
  WHERE DEPT = 'A00'
  ORDER BY JOB, EDL;
```

The result table looks like the following example:

```
JOB     EDL     LASTNAME
====    ===     ==========
PRES    18      HAAS
SLS     14      CONNOR
SLS     18      HEMMINGER
```

GUPI

Related reference

order-by-clause (Db2 SQL)

Sort keys with expressions

You can specify an expression with operators as the sort key for the result table of a SELECT statement.

When you specify an expression with operators as the sort key, the query to which ordering is applied must be a subselect.

GUPI

Example: The following query is a part of a subselect. The query retrieves the employee numbers, salaries, commissions, and total compensation (salary plus commission) for employees with a total compensation greater than 40000. Order the results by total compensation:

```
SELECT EMPNO, SALARY, COMM, SALARY+COMM AS "TOTAL COMP"
  FROM EMP
  WHERE SALARY+COMM> 40000
  ORDER BY SALARY+COMM;
```

The result table looks like the following example:

```
EMPNO    SALARY     COMM      TOTAL COMP
======   ========   =======   ==========
000030   38250.00   3060.00   41310.00
000020   41250.00   3300.00   44550.00
200010   46500.00   4220.00   50720.00
000010   52750.00   4220.00   56970.00
```

GUPI

Related reference

order-by-clause (Db2 SQL)

Ways to summarize group values

You can use the GROUP BY clause to summarize group values.

Use GROUP BY to group rows by the values of one or more columns. You can then apply aggregate functions to each group. You can use an expression in the GROUP BY clause to specify how to group the rows.

Except for the columns that are named in the GROUP BY clause, the SELECT statement must specify any other selected columns as an operand of one of the aggregate functions.

Example: This query lists, for each department, the lowest and highest education level within that department: The result table looks like this:

```
SELECT DEPT, MIN(EDL), MAX(EDL)
  FROM EMP
  GROUP BY DEPT;
```

```
DEPT
====     ==     ==
A00      14     18
B01      18     18
```

```
C01      18     20
D11      16     18
E21      14     16
```

If a column that you specify in the GROUP BY clause contains null values, Db2 considers those null values to be equal, and all nulls form a single group.

Within the SELECT statement, the GROUP BY clause follows the FROM clause and any WHERE clause, and it precedes the HAVING and ORDER BY clauses.

You can also group the rows by the values of more than one column.

Example: This query finds the average salary for employees with the same job in departments D11 and E21:

```
SELECT DEPT, JOB, AVG(SALARY) AS AVG_SALARY
  FROM EMP
  WHERE DEPT IN ('D11', 'E21')
  GROUP BY DEPT, JOB;
```

The result table looks like this:

```
DEPT     JOB         AVG_SALARY
====     ===     ==============
D11      DES     28790.00000000
D11      MGR     32250.00000000
E21      FLD     23053.33333333
```

In this example, Db2 groups the rows first by department number and next (within each department) by job before deriving the average salary value for each group.

Example: This query finds the average salary for all employees that were hired in the same year. You can use the following subselect to group the rows by the year of hire:

```
SELECT AVG(SALARY), YEAR(HIREDATE)
  FROM EMP
  GROUP BY YEAR(HIREDATE);
```

Related tasks

Coding SQL statements to avoid unnecessary processing (Db2 Performance)

Related reference

group-by-clause (Db2 SQL)

order-by-clause (Db2 SQL)

select-statement (Db2 SQL)

Ways to merge lists of values

There are several ways to use the UNION keyword for merging lists of values.

A *union* is an SQL operation that combines the results of two SELECT statements to form a single result table. When Db2 encounters the UNION keyword, it processes each SELECT statement to form an *interim result table*. Db2 then combines the interim result table of each statement. If you use UNION to combine two columns with the same name, the corresponding column of the result table inherits that name.

You can use the UNION keyword to obtain distinct rows in the result table of a union, or you can use UNION with the optional keyword ALL to obtain all rows, including duplicates.

How to eliminate duplicates

Use UNION to eliminate duplicates when merging lists of values that are obtained from several tables. The following example combines values from the EMP table and the EMPPROJACT table.

Example 1: List the employee numbers of all employees for which either of the following statements is true:

- The department number of the employee begins with 'D'.

- The employee is assigned to projects whose project numbers begin with 'MA'.

```
SELECT EMPNO FROM EMP
  WHERE DEPT LIKE 'D%'
  UNION
  SELECT EMPNO FROM EMPPROJACT
  WHERE PROJNO LIKE 'MA
```

The result table looks like the following example:

```
EMPNO
======
000010
000020
000060
000200
000220
```

The result is the union of two result tables, one formed from the EMP table, the other formed from the EMPPROJACT table. The result, a one-column table, is a list of employee numbers. The entries in the list are distinct.

How to retain duplicates

If you want to keep duplicates in the result of a union, specify the optional keyword ALL after the UNION keyword.

Example 1: Replace the UNION keyword in the previous example with UNION ALL:

```
SELECT EMPNO FROM EMP
  WHERE DEPT LIKE 'D%'
  UNION ALL
  SELECT EMPNO FROM EMPPROJACT
  WHERE PROJNO LIKE 'MA
```

The result table looks like the following example:

```
EMPNO
======
000220
000200
000060
000010
000020
000010
```

Now, 000010 is included in the list more than once because this employee works in a department that begins with 'D' and also works on a project that begins with 'MA'.

Related reference

fullselect (Db2 SQL)

Ways to specify search conditions

You can use the HAVING clause in a variety of ways to specify search conditions.

Use HAVING to specify a search condition that each retrieved group must satisfy. The HAVING clause acts like a WHERE clause for groups, and it can contain the same kind of search conditions that you can specify in a WHERE clause. The search condition in the HAVING clause tests properties of each group rather than properties of individual rows in the group.

GUPI

Example: Consider this query:

```
SELECT DEPT, AVG(SALARY) AS AVG_SALARY
  FROM EMP
  GROUP BY DEPT
```

```
  HAVING COUNT(*)> 1
  ORDER BY DEPT;
```

The result table looks like this:

```
DEPT         AVG_SALARY
====     ==============
A00      42833.33333333
C01      31696.66666666
D11      29943.33333333
E21      23053.33333333
```

The HAVING COUNT(*)> 1 clause ensures that only departments with more than one member are displayed. (In this case, department B01 is not displayed because it consists of only one employee.)

GUPI

Example: You can use the HAVING clause to retrieve the average salary and minimum education level of employees that were hired after 1990 and who report to departments in which the education level of all employees is greater than or equal to 14. Assuming that you want results only from departments A00 and D11, the following SQL statement tests the group property, MIN(EDL): GUPI

```
SELECT DEPT, AVG(SALARY) AS AVG_SALARY,
  MIN(EDL) AS MIN_EDL
  FROM EMP
  WHERE HIREDATE>= '1990-01-01' AND DEPT IN ('A00', 'D11')
  GROUP BY DEPT
  HAVING MIN(EDL)>= 14;
```

The result table looks like this:

```
DEPT         AVG_SALARY     MIN_EDL
====     ==============     =======
A00      29250.00000000          14
D11      29943.33333333          16
```

GUPI

When you specify both GROUP BY and HAVING, the HAVING clause must follow the GROUP BY clause in the syntax. A function in a HAVING clause can include multiple occurrences of the DISTINCT clause. You can also connect multiple predicates in a HAVING clause with AND and OR, and you can use NOT for any predicate of a search condition.

Related concepts

Ways to summarize group values
You can use the GROUP BY clause to summarize group values.

Predicates (Db2 SQL)

Related reference

having-clause (Db2 SQL)

select-statement (Db2 SQL)

where-clause (Db2 SQL)

Ways to join data from more than one table

When you want to see information from multiple tables, you can use a SELECT statement. SELECT statements can retrieve and join column values from two or more tables into a single row. The retrieval is based on a specified condition, typically of matching column values.

The main ingredient of a join is, typically, matching column values in rows of each table that participates in the join. The result of a join associates rows from one table with rows from another table. Depending on the type of join operation, some rows might be formed that contain column values in one table that do not match column values in another table.

A *joined-table* specifies an intermediate result table that is the result of either an inner join or an outer join. The table is derived by applying one of the join operators—INNER, FULL OUTER, LEFT OUTER, or RIGHT OUTER—to its operands.

Db2 supports inner joins and outer joins (left, right, and full).

Db2 supports inner joins and outer joins (left, right, and full).

Inner join
Combines each row of the left table with each row of the right table, keeping only the rows in which the join condition is true.

Outer join
Includes the rows that are produced by the inner join, plus the missing rows, depending on the type of outer join:

Left outer join
Includes the rows from the left table that were missing from the inner join.

Right outer join
Includes the rows from the right table that were missing from the inner join.

Full outer join
Includes the rows from both tables that were missing from the inner join.

The majority of examples in this topic use two example tables: the parts table (PARTS) and the products table (PRODUCTS), which consist of hardware supplies.

The following figure shows that each row in the PARTS table contains data for a single part: the part name, the part number, and the supplier of the part.

PARTS

PART	PROD#	SUPPLIER
WIRE	10	ACWF
OIL	160	WESTERN_CHEM
MAGNETS	10	BATEMAN
PLASTIC	30	PLASTIK_CORP
BLADES	205	ACE_STEEL

Figure 12. Example PARTS table

The following figure shows that each row in the PRODUCTS table contains data for a single product: the product number, name, and price.

PRODUCTS

PROD#	PRODUCT	PRICE
505	SCREWDRIVER	3.70
30	RELAY	7.55
205	SAW	18.90
10	GENERATOR	45.75

Figure 13. Example PRODUCTS table

The following figure shows the ways to combine the PARTS and PRODUCTS tables by using outer join functions. The illustration is based on a subset of columns in each table.

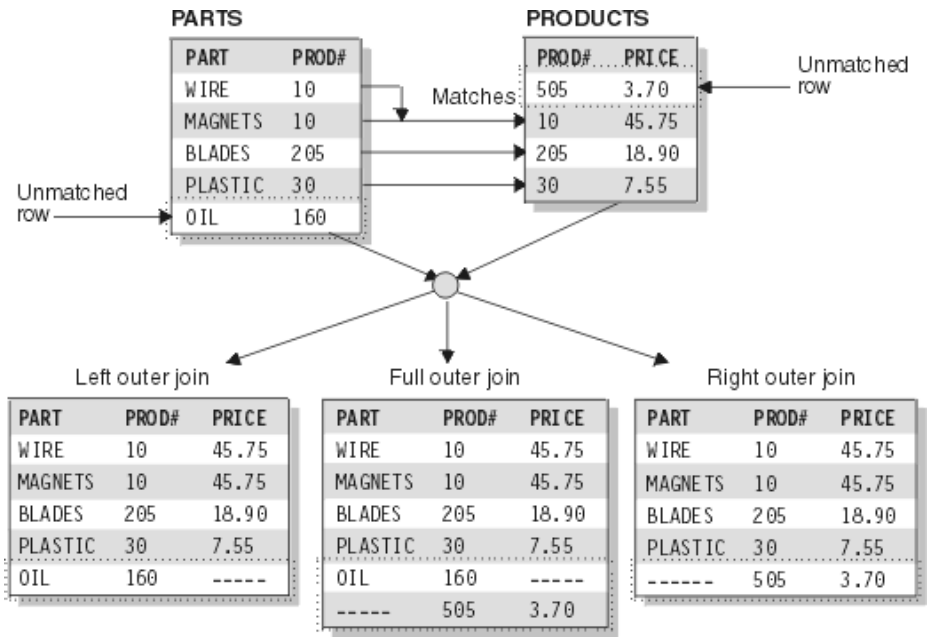


Figure 14. Outer joins of two tables

An inner join consists of rows that are formed from the PARTS and PRODUCTS tables, based on matching the equality of column values between the PROD# column in the PARTS table and the PROD# column in the PRODUCTS table. The inner join does not contain any rows that are formed from unmatched columns when the PROD# columns are not equal.

You can specify joins in the FROM clause of a query. Data from the rows that satisfy the search conditions are joined from all the tables to form the result table.

The result columns of a join have names if the outermost SELECT list refers to base columns. However, if you use a function (such as COALESCE) to build a column of the result, that column does not have a name unless you use the AS clause in the SELECT list.

Related concepts

Investigating join operations (Db2 Performance)

Related tasks

Joining data from more than one table (Db2 Application programming and SQL)

Related reference

select-statement (Db2 SQL)

from-clause (Db2 SQL)

joined-table (Db2 SQL)

Inner join

You can use an inner join in a SELECT statement to retrieve only the rows that satisfy the join conditions on every specified table.

You can request an inner join, by running a SELECT statement in which you specify the tables that you want to join the FROM clause and specify a WHERE clause or an ON clause to indicate the join condition. The join condition can be any simple or compound search condition that does not contain a subquery reference.

In the simplest type of inner join, the join condition is *column1=column2*.

GUPI

Example: You can join the PARTS and PRODUCTS tables on the PROD# column to form a table of parts with their suppliers and the products that use the parts. Consider the two following SELECT statements:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS, PRODUCTS
  WHERE PARTS.PROD# = PRODUCTS.PROD#;
```

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
  ON PARTS.PROD# = PRODUCTS.PROD#;
```

Either of these statements gives this result:

```
PART        SUPPLIER         PROD#     PRODUCT
=======     ============     =====     =========
WIRE        ACWF             10        GENERATOR
MAGNETS     BATEMAN          10        GENERATOR
BLADES      ACE_STEEL        205       SAW
PLASTIC     PLASTIK_CORP     30        RELAY
```

GUPI

Notice three things about this example:

- One part in the PARTS table (OIL) has a product number (160) that is not in the PRODUCTS table. One product (505, SCREWDRIVER) has no parts listed in the PARTS table. OIL and SCREWDRIVER do not appear in the result of the join.
- Explicit syntax expresses that this join is an inner join. You can use INNER JOIN in the FROM clause instead of the comma. ON (rather than WHERE) specifies the join condition when you explicitly join tables in the FROM clause.
- If you do not specify a WHERE clause in the first form of the query, the result table contains all possible combinations of rows for the tables that are identified in the FROM clause. You can obtain the same result by specifying a join condition that is always true in the second form of the query. GUPI

  Example: Consider this query:

  ```
  SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
    FROM PARTS INNER JOIN PRODUCTS
    ON 1=1;
  ```

  The number of rows in the result table is the product of the number of rows in each table:

  ```
  PART        SUPPLIER         PROD#     PRODUCT
  =======     ============     =====     ===========
  WIRE        ACWF             10        SCREWDRIVER
  WIRE        ACWF             10        RELAY
  WIRE        ACWF             10        SAW
  WIRE        ACWF             10        GENERATOR
  OIL         WESTERN_CHEM     160       SCREWDRIVER
  OIL         WESTERN_CHEM     160       RELAY
  OIL         WESTERN_CHEM     160       SAW
  OIL         WESTERN_CHEM     160       GENERATOR
  ⋮
  ```

GUPI

You can specify more complicated join conditions to obtain different sets of results.

GUPI

Example: To eliminate the suppliers that begin with the letter A from the table of parts, suppliers, product numbers, and products, write a query like the following example:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#
    AND SUPPLIER NOT LIKE 'A%';
```

The result of the query is all rows that do not have a supplier that begins with A:

```
PART        SUPPLIER       PROD#     PRODUCT
=======     ============   =====     =========
MAGNETS     BATEMAN        10        GENERATOR
PLASTIC     PLASTIK_CORP   30        RELAY
```

Example: This example joins the PROJ table to itself by using an inner join. The query returns the number and name of each major project, followed by the number and name of the project that is part of it:

```
SELECT A.PROJNO AS "MAJOR PROJ",
       A.PROJNAME AS "MAJOR PROJ NAME",
       B.PROJNO AS "PROJ #",
       B.PROJNAME AS "PROJ NAME"
  FROM PROJ A, PROJ B
  WHERE A.PROJNO=B.MAJPROJ;
```

In this example, A indicates the first instance of table PROJ, and B indicates a second instance of this table. The join condition is such that the value in column PROJNO in table PROJ A must be equal to a value in column MAJPROJ in table PROJ B.

The result table looks like the following example:

```
MAJOR PROJ  MAJOR PROJ NAME        PROJ #     PROJ NAME
==========  ===============        ======     ====================
AD3100      ADMIN SERVICES         AD3110     GENERAL AD SYSTEMS
AD3110      GENERAL AD SYSTEMS     AD3111     PAYROLL PROGRAMMING
AD3110      GENERAL AD SYSTEMS     AD3112     PERSONNEL PROGRAMMG
AD3110      GENERAL AD SYSTEMS     AD3113     ACCOUNT.PROGRAMMING
MA2100      WELD LINE AUTOMATION   MA2110     W L PROGRAMMING
MA2110      W L PROGRAMMING        MA2111     W L PROGRAM DESIGN
MA2110      W L PROGRAMMING        MA2112     W L ROBOT DESIGN
MA2110      W L PROGRAMMING        MA2113     W L PROD CONT PROGS
OP1000      OPERATION SUPPORT      OP1010     OPERATION
OP2000      GEN SYSTEMS SERVICES   OP2010     SYSTEMS SUPPORT
OP2010      SYSTEMS SUPPORT        OP2011     SCP SYSTEMS SUPPORT
OP2010      SYSTEMS SUPPORT        OP2012     APPLICATIONS SUPPORT
OP2010      SYSTEMS SUPPORT        OP2013     DB/DC SUPPORT
MA2100      WELD LINE AUTOMATION   PL2100     WELD LINE PLANNING
```

In this example, the comma in the FROM clause implicitly specifies an inner join, and it acts the same as if the INNER JOIN keywords had been used. When you use the comma for an inner join, you must specify the join condition in the WHERE clause. When you use the INNER JOIN keywords, you must specify the join condition in the ON clause.

GUPI

Related concepts

Accessing Db2 data

You can retrieve data by using the SQL statement SELECT to specify a result table that can be derived from one or more tables.

Subqueries

You can use a subquery to narrow a search condition that is based on information in an interim table.

Inner joins (Db2 Application programming and SQL)

Related reference

select-statement (Db2 SQL)

join-condition (Db2 SQL)

Left outer join

The LEFT OUTER JOIN clause lists rows from the left table even if there are no matching rows on right table.

As in an inner join, the join condition of a left outer join can be any simple or compound search condition that does not contain a subquery reference.

GUPI

Example: To include rows from the PARTS table that have no matching values in the PRODUCTS table and to include prices that exceed $10.00, run this query:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT, PRICE
  FROM PARTS LEFT OUTER JOIN PRODUCTS
  ON PARTS.PROD#=PRODUCTS.PROD#
  AND PRODUCTS.PRICE>10.00;
```

The result table looks like the following example:

```
PART        SUPPLIER         PROD#     PRODUCT       PRICE
=======     ============     =====     =========     =====
WIRE        ACWF             10        GENERATOR     45.75
MAGNETS     BATEMAN          10        GENERATOR     45.75
OIL         WESTERN_CHEM     160       ---------     -----
BLADES      ACE_STEEL        205       SAW           18.90
PLASTIC     PLASTIK_CORP     30        ---------     -----
```

Because the PARTS table can have rows that are not matched by values in the joined columns and because the PRICE column is not in the PARTS table, rows in which the PRICE value does not exceed $10.00 are included in the result of the join, but the PRICE value is set to null.

In this result table, the row for PROD# 160 has null values on the right two columns because PROD# 160 does not match another product number. PROD# 30 has null values on the right two columns because the price of PROD# 30 is less than $10.00.

GUPI

Related concepts

Subqueries

You can use a subquery to narrow a search condition that is based on information in an interim table.

Left outer join (Db2 Application programming and SQL)

Outer joins (Db2 Application programming and SQL)

Related reference

select-statement (Db2 SQL)

join-condition (Db2 SQL)

Right outer join

The RIGHT OUTER JOIN clause lists rows from the right table even if there are no matching rows on left table.

As in an inner join, the join condition of a right outer join can be any simple or compound search condition that does not contain a subquery reference.

GUPI

Example: To include rows from the PRODUCTS table that have no matching values in the PARTS table and to include only prices that exceed $10.00, run this query:

```
SELECT PART, SUPPLIER, PRODUCTS.PROD#, PRODUCT, PRICE
  FROM PARTS RIGHT OUTER JOIN PRODUCTS
  ON PARTS.PROD# = PRODUCTS.PROD#
  WHERE PRODUCTS.PRICE>10.00;
```

The result table looks like the following example:

```
PART        SUPPLIER        PROD#     PRODUCT         PRICE
=======     ============    =====     ==========      =====
MAGNETS     BATEMAN         10        GENERATOR       45.75
WIRE        ACWF            10        GENERATOR       45.75
BLADES      ACE_STEEL       205       SAW             18.90
```

Because the PRODUCTS table cannot have rows that are not matched by values in the joined columns and because the PRICE column is in the PRODUCTS table, rows in which the PRICE value does not exceed $10.00 are not included in the result of the join.

GUPI

Related concepts

Right outer join (Db2 Application programming and SQL)

Outer joins (Db2 Application programming and SQL)

Related reference

select-statement (Db2 SQL)

join-condition (Db2 SQL)

Full outer join

The FULL OUTER JOIN clause results in the inclusion of rows from two tables. If a value is missing when rows are joined, that value is null in the result table.

The join condition for a full outer join must be a search condition that compares two columns. The predicates of the search condition can be combined only with AND. Each predicate must have the form 'expression = expression'.

GUPI

Example 1: This query performs a full outer join of the PARTS and PRODUCTS tables:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
  ON PARTS.PROD# = PRODUCTS.PROD#;
```

The result table looks like this:

```
PART         SUPPLIER       PROD#     PRODUCT
========     ============   =====     ==========
WIRE         ACWF           10        GENERATOR
MAGNETS      BATEMAN        10        GENERATOR
OIL          WESTERN_CHEM   160       -----------
BLADES       ACE_STEEL      205       SAW
PLASTIC      PLASTIK_CORP   30        RELAY
-------      ------------   -----     SCREWDRIVER
```

GUPI

Using COALESCE

This function can be particularly useful in full outer join operations because it returns the first nonnull value. For example, notice that the result in the example above is null for SCREWDRIVER, even though the PRODUCTS table contains a product number for SCREWDRIVER. If you select PRODUCTS.PROD# instead, PROD# is null for OIL. If you select both PRODUCTS.PROD# and PARTS.PROD#, the result contains two columns, and both columns contain some null values.

GUPI

Example 2: You can merge data from both columns into a single column, eliminating the null values, by using the COALESCE function. Consider this query with the same PARTS and PRODUCTS tables:

```
SELECT PART, SUPPLIER,
  COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM, PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
  ON PARTS.PROD# = PRODUCTS.PROD#;
```

This statement gives this result:

```
PART        SUPPLIER         PRODNUM     PRODUCT
=======     ============     =======     ==========
WIRE        ACWF             10          GENERATOR
MAGNETS     BATEMAN          10          GENERATOR
OIL         WESTERN_CHEM     160         -----------
BLADES      ACE_STEEL        205         SAW
PLASTIC     PLASTIK_CORP     30          RELAY
-------     ------------     505         SCREWDRIVER
```

The AS clause AS PRODNUM provides a name for the result of the COALESCE function.

GUPI

Related concepts

Full outer join (Db2 Application programming and SQL)

Related reference

select-statement (Db2 SQL)

join-condition (Db2 SQL)

Subqueries

You can use a subquery to narrow a search condition that is based on information in an interim table.

A *subquery* is a nested SQL statement, or *subselect*, that contains a SELECT statement within the WHERE or HAVING clause of another SQL statement. You can also code more complex subqueries, such as correlated subqueries and subqueries with quantified predicates.

You can use a subquery when you need to narrow your search condition that is based on information in an interim table. For example, you might want to find all employee numbers in one table that also exist for a given project in a second table.

GUPI

Example: Suppose that you want a list of the employee numbers, names, and commissions of all employees that work on a particular project, such as project number IF2000. The first part of the SELECT statement is easy to write:

```
SELECT EMPNO, LASTNAME, COMM
  FROM EMP
  WHERE EMPNO
⋮
```

GUPI

However, you cannot go further because the EMP table does not include project number data. You do not know which employees are working on project IF2000 without issuing another SELECT statement against the EMPPROJACT table.

You can use a subselect to solve this problem. The SELECT statement that surrounds the subquery is the outer SELECT.

GUPI

Example: This query expands the SELECT statement that started in the previous example to include a subquery:

```
SELECT EMPNO, LASTNAME, COMM
  FROM EMP
  WHERE EMPNO IN
    (SELECT EMPNO
       FROM EMPPROJACT
       WHERE PROJNO = 'IF2000');
```

To better understand what happens as a result from this SQL statement, imagine that Db2 goes through the following process:

1. Db2 evaluates the subquery to obtain a list of EMPNO values:

   ```
   (SELECT EMPNO
      FROM EMPPROJACT
      WHERE PROJNO = 'IF2000');
   ```

   The result is the following interim result table:

   ```
   EMPNO
   ======
   ```

```
000140
000140
000030
```

2. The interim result table then serves as a list in the search condition of the outer SELECT. Effectively, Db2 runs this SELECT statement:

```
SELECT EMPNO, LASTNAME, COMM
  FROM EMP
  WHERE EMPNO IN
  ('000140', '000030');
```

The result table looks like this:

```
EMPNO     LASTNAME        COMM
=====     ========     =======
000140    NICHOLLS     2274.00
000030    KWAN         3060.00
```

GUPI

Related concepts

Subqueries (Db2 Application programming and SQL)

Related tasks

Writing efficient subqueries (Db2 Performance)

Related reference

subselect (Db2 SQL)

Examples of subselects (Db2 SQL)

Ways to access Db2 data that is not in a table

You can access Db2 data even if it is not in a table.

This method of data access can be accomplished in two ways.

GUPI

- Set the contents of a host variable to the value of an expression by using the SET host-variable assignment statement.

  Example:

```
EXEC SQL SET :HVRANDVAL = RAND(:HVRAND);
```

- In addition, you can use the VALUES INTO statement to return the value of an expression in a host variable.

  Example:

```
EXEC SQL VALUES RAND(:HVRAND)
INTO :HVRANDVAL;
```

GUPI

Related concepts

Data access with host variables

You can use host variables, host variable arrays, and host structures in your application program to exchange data between the application and the DBMS.

Modifying Db2 data

You can use SQL statements to add, modify, merge, and remove data in existing tables. You can use the INSERT, UPDATE, MERGE, TRUNCATE, and DELETE statements to manipulate Db2 data.

If you insert, update, merge, or delete data, you can retrieve the data immediately. If you open a cursor and then modify data, you see the modified data only in some circumstances.

Any modifications must maintain the integrity of table relationships. Db2 ensures that an insert, update, or delete operation does not violate any referential constraint or check constraint that is defined on a table.

Before modifying data in your tables, create duplicate tables for testing purposes so that the original table data remains intact. Assume that you created two new tables, NEWDEPT and NEWEMP, that duplicate the DEPT and EMP tables.

Related concepts

Use of check constraints to enforce validity of column values
You can use check constraints to ensure that only values from the domain for the column or attribute are allowed.

Related tasks

Adding and modifying data from application programs (Db2 Application programming and SQL)

Related reference

INSERT (Db2 SQL)
UPDATE (Db2 SQL)
MERGE (Db2 SQL)
TRUNCATE (Db2 SQL)
DELETE (Db2 SQL)

Insert statements

You can use an INSERT statement to add new rows to a table or view.

You can use an INSERT statement to take the following actions:

- Specify the values to insert in a single row. You can specify constants, host variables, expressions, DEFAULT, or NULL.
- Use host variable arrays in the VALUES clause of the INSERT FOR *n* ROWS statement to insert multiple rows into a table.
- Include a SELECT statement in the INSERT statement to tell Db2 that another table or view contains the data for the new row or rows.

You can add new data to an existing table in other ways, too. You might need to add large amounts of data to an existing table. Some efficient options include copying a table into another table, writing an application program that enters data into a table, and using the Db2 LOAD utility to enter data.

GUPI

Suppose that you want to add a new row to the NEWDEPT table. Use this INSERT statement:

```
INSERT INTO NEWDEPT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT)
  VALUES ('E31', 'PUBLISHING', '000020', 'D11');
```

After inserting the new department row into the NEWDEPT table, you can use a SELECT statement to see what the modified table looks like. Use this query:

```
SELECT *
  FROM NEWDEPT
  WHERE DEPTNO LIKE 'E%'
  ORDER BY DEPTNO;
```

The result table gives you the new department row that you inserted for department E31 and the existing departments with a department number beginning in E.

```
DEPTNO     DEPTNAME             MGRNO      ADMRDEPT
======     ================     ======     ========
E21        SOFTWARE SUPPORT     ------          D11
E31        PUBLISHING           000020          D11
```

GUPI

Related concepts

Programming for Db2 for z/OS

You use application programs that contain SQL statements to perform data definition or data manipulation operations on Db2 for z/OS objects.

Related tasks

Inserting data into tables (Db2 Application programming and SQL)

Related reference

INSERT (Db2 SQL)

where-clause (Db2 SQL)

Update statements

You can change the data in a table by using the UPDATE statement or the MERGE statement.

The UPDATE statement modifies zero or more rows of a table, depending on how many rows satisfy the search condition that you specify in the WHERE clause.

You can use an UPDATE or MERGE statement to specify the values that are to be updated in a single row. You can specify constants, host variables, expressions, DEFAULT, or NULL. Specify NULL to remove a value from a row's column (without removing the row).

GUPI

Suppose that an employee gets a promotion. To update several items of the employee's data in the NEWEMP table that reflects the move, use this UPDATE statement:

```
UPDATE NEWEMP
  SET JOB = 'MGR',
  DEPT = 'E21'
  WHERE EMPNO = '100125';
```

GUPI

Related tasks

Updating table data (Db2 Application programming and SQL)

Related reference

UPDATE (Db2 SQL)

where-clause (Db2 SQL)

Merge statements

The MERGE statement updates a target with specified input data.

The target of a MERGE statement can be a table or a view. Rows in the target that match the input data are updated as specified, and rows that do not exist in the target are inserted. You also can use a MERGE statement with host variable arrays to insert and update data. The MERGE statement can also update underlying tables or views of a fullselect.

Related tasks

Inserting data and updating data in a single operation (Db2 Application programming and SQL)

Related reference

MERGE (Db2 SQL)

where-clause (Db2 SQL)

Delete statements

You can use the DELETE statement to remove entire rows from a table.

The DELETE statement removes zero or more rows of a table, depending on how many rows satisfy the search condition that you specify in the WHERE clause. If you omit a WHERE clause from a DELETE statement, Db2 removes all rows from the table or view you name. Therefore, use the DELETE statement carefully. The DELETE statement does not remove specific columns from the row.

GUPI

Consider this DELETE statement:

```
DELETE FROM NEWEMP
  WHERE EMPNO = '000060';
```

This DELETE statement deletes each row in the NEWEMP table that has employee number 000060.

GUPI

Related tasks

Deleting data from tables (Db2 Application programming and SQL)

Related reference

MERGE (Db2 SQL)

where-clause (Db2 SQL)

Truncate statements

You can use the TRUNCATE statement to delete all rows for base tables or declared global temporary tables.

You can embed a TRUNCATE statement in an application program or issue it interactively. TRUNCATE statements are executable statements that you can prepare dynamically. To truncate a table, you must have the proper authorization or be the owner of the table. The TRUNCATE statement must not be confused with the TRUNCATE function.

GUPI

This example empties an unused inventory table regardless of any existing triggers and returns its allocated space.

```
   TRUNCATE TABLE INVENTORY
            DROP STORAGE
            IGNORE DELETE TRIGGERS;
```

GUPI

Related tasks

Deleting data from tables (Db2 Application programming and SQL)

Related reference

TRUNCATE (Db2 SQL)

where-clause (Db2 SQL)

Defining Db2 objects

You can use SQL statements to create, modify, and drop Db2 objects, by using various CREATE, ALTER and DROP statements.

Creation of tables

Designing tables that many applications use is a critical task. Table design can be difficult because you can represent the same information in many different ways. This information briefly describes how tables are created and altered, and how authorization is controlled.

You create tables by using the SQL CREATE TABLE statement. At some point after you create and start using your tables, you might need to make changes to them. The ALTER TABLE statement lets you add and change columns, add or drop a primary key or foreign key, add or drop table check constraints, or add and change partitions. Carefully consider design changes to avoid or reduce the disruption to your applications.

If you have DBADM (database administration) authority, you probably want to control the creation of Db2 databases and table spaces. These objects can have a big impact on the performance, storage, and

security of the entire relational database. In some cases, you also want to retain the responsibility for creating tables. After designing the relational database, you can create the necessary tables for application programs. You can then pass the authorization for their use to the application developers, either directly or indirectly, by using views.

However, if you want to, you can grant the authority for creating tables to those who are responsible for implementing the application. For example, you probably want to authorize certain application programmers to create tables if they need temporary tables for testing purposes.

Some users in your organization might want to use Db2 with minimum assistance or control. You can define a separate storage group and database for these users and authorize them to create whatever data objects they need, such as tables.

Related concepts

Authorization and security mechanisms for data access
Authorization is an important part of controlling Db2. The security and authorization mechanisms that control access to Db2 data are both direct and indirect.

Db2 objects and their relationships
Logical data modeling and physical data modeling are two tasks that you need to perform to design a Db2 database.

Related tasks

Creating tables (Db2 Application programming and SQL)

Related reference

CREATE TABLE (Db2 SQL)

ALTER TABLE (Db2 SQL)

Related information

Implementing Db2 tables (Db2 Administration Guide)

Types of tables

In Db2, you store user data in tables. Db2 supports several types of tables, each of which has its own purpose and characteristics.

Db2 supports the following types of tables:

accelerator-only table
A table that stores rows only in the accelerator, not in Db2. The table and column definition of the accelerator-only table is contained in Db2 catalog tables. Any queries that reference the accelerator-only table, must be executed in the accelerator. If a query that references an accelerator-only table is not eligible for query acceleration, an error is issued. To change the contents of an accelerator-only table, the data change statement must be executed in the accelerator.

archive table
A table that stores rows that are deleted from another table.

archive-enabled table
A table that has an associated archive table. When rows are deleted from an archive-enabled table, Db2 can automatically insert those rows into an archive table.

auxiliary table
A table created with the SQL statement CREATE AUXILIARY TABLE and used to hold the data for a column that is defined in a base table.

base table
The most common type of table in Db2. You create a base table with the SQL CREATE TABLE statement. The Db2 catalog table, SYSIBM.SYSTABLES, stores the description of the base table. The table description and table data are persistent. All programs and users that refer to this type of table refer to the same description of the table and to the same instance of the table.

clone table
A table that is structurally identical to a base table. You create a clone table by using an ALTER TABLE statement for the base table that includes an ADD CLONE clause. The clone table is created in a

different instance of the same table space as the base table, is structurally identical to the base table in every way, and has the same indexes, before triggers, and LOB objects. In the Db2 catalog, the SYSTABLESPACE table indicates that the table space has only one table in it, but SYSTABLESPACE.CLONE indicates that a clone table exists. Clone tables can be created only in a partition-by range or partition-by-growth table space that is managed by Db2. The base and clone table each have separate underlying VSAM data sets (identified by their data set instance numbers) that contain independent rows of data.

empty table
A table with zero rows.

history table
A table that is used to store historical versions of rows from the associated system-period temporal table.

materialized query table
A table, which you define with the SQL CREATE TABLE statement, that contains materialized data that is derived from one or more source tables. Materialized query tables are useful for complex queries that run on large amounts of data. Db2 can precompute all or part of such queries and use the precomputed, or materialized, results to answer the queries more efficiently. Materialized query tables are commonly used in data warehousing and business intelligence applications.

Several Db2 catalog tables, including SYSIBM.SYSTABLES and SYSIBM.SYSVIEWS, store the description of the materialized query table and information about its dependency on a table, view, or function. The attributes that define a materialized query table tell Db2 whether the table is:

- System-maintained or user-maintained.
- Refreshable: All materialized tables can be updated with the REFRESH TABLE statement. Only user-maintained materialized query tables can also be updated with the LOAD utility and the UPDATE, INSERT, and DELETE SQL statements.
- Enabled for query optimization: You can enable or disable the use of a materialized query table in automatic query rewrite.

Materialized query tables can be used to improve the performance of dynamic SQL queries. If Db2 determines that a portion of a query could be resolved using a materialized query table, the query might be rewritten by Db2 to use the materialized query table. This decision is based in part on the settings of the CURRENT REFRESH AGE and the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers.

result table
A table that contains a set of rows that Db2 selects or generates, directly or indirectly, from one or more base tables in response to an SQL statement. Unlike a base table or a temporary table, a result table is not an object that you define using a CREATE statement.

sample table
One of several tables shipped with the Db2 licensed program that contains sample data. Many examples in this information are based on sample tables.

temporal table
A table that records the period of time when a row is valid.

Db2 supports two types of periods, which are the system period (SYSTEM_TIME) and the application period (BUSINESS_TIME). The system period consists of a pair of columns with system-maintained values that indicates the period of time when a row is valid. The application period consists of a pair of columns with application-maintained values that indicates the period of time when a row is valid.

system-period temporal table
A system-period temporal table is a base table that is defined with system-period data versioning. You can modify an existing table to become a system-period temporal table by specifying the ADD PERIOD SYSTEM_TIME clause on the ALTER TABLE statement. After creating a history table that corresponds to the system-period temporal table, you can define system-period data versioning on the table by issuing the ALTER TABLE ADD VERSIONING statement with the USE HISTORY table clause.

application-period temporal table
An application-period temporal table is a base table that includes an application period (BUSINESS_TIME). You can modify an existing table to become an application-period temporal table by specifying the ADD PERIOD BUSINESS_TIME clause on the ALTER TABLE statement.

bitemporal table
A bitemporal table is a table that is both a system-period temporal table and an application-period temporal table. You can use a bitemporal table to keep application period information and system-based historical information. Therefore, you have a lot of flexibility in how you query data, based on periods of time.

temporary table
A table that is defined by the SQL statement CREATE GLOBAL TEMPORARY TABLE or DECLARE GLOBAL TEMPORARY TABLE to hold data temporarily. Temporary tables are especially useful when you need to sort or query intermediate result tables that contain many rows, but you want to store only a small subset of those rows permanently.

created global temporary table
A table that you define with the SQL CREATE GLOBAL TEMPORARY TABLE statement. The Db2 catalog table, SYSIBM.SYSTABLES, stores the description of the created temporary table. The description of the table is persistent and shareable. However, each individual application process that refers to a created temporary table has its own distinct instance of the table. That is, if application process A and application process B both use a created temporary table named TEMPTAB:

- Each application process uses the same table description.
- Neither application process has access to or knowledge of the rows in the other application instance of TEMPTAB.

declared global temporary table
A table that you define with the SQL DECLARE GLOBAL TEMPORARY TABLE statement. The Db2 catalog does not store a description of the declared temporary table. Therefore, the description and the instance of the table are not persistent. Multiple application processes can refer to the same declared temporary table by name, but they do not actually share the same description or instance of the table. For example, assume that application process A defines a declared temporary table named TEMP1 with 15 columns. Application process B defines a declared temporary table named TEMP1 with five columns. Each application process uses its own description of TEMP1; neither application process has access to or knowledge of rows in the other application instance of TEMP1.

XML table
A special table that holds only XML data. When you create a table with an XML column, Db2 implicitly creates an XML table space and an XML table to store the XML data.

These different types of tables differ in other ways that this topic does not describe.

Related concepts

Creation of large objects
Defining large objects to Db2 is different than defining other types of data and objects.

Creation of materialized query tables
Materialized query tables improve the performance of complex queries that operate on large amounts of data.

Db2 catalog
Db2 maintains a set of tables that contain information about the data that Db2 controls. These tables are collectively known as the *catalog*.

Archive-enabled tables and archive tables
If you have a table that contains a significant amount of historical data that is not often referenced, consider creating archive tables. An *archive table* is a table that stores older rows from another table.

Related reference

Db2 sample tables

Much of the Db2 information refers to or relies on the Db2 sample tables. As a group, the tables include information that describes employees, departments, projects, and activities, and they make up a sample application that exemplifies many of the features of Db2.

Archive-enabled tables and archive tables

If you have a table that contains a significant amount of historical data that is not often referenced, consider creating archive tables. An *archive table* is a table that stores older rows from another table.

The original table is called an *archive-enabled table*. Db2 can automatically store rows that are deleted from an archive-enabled table in an associated archive table. When you query an archive-enabled table, you can specify whether you want those queries to include data from the archive table.

Archive tables have the following advantages:

- Db2 can manage historical data for you. You do not have to manually move data to a separate table.
- Because rows that are infrequently accessed are stored in a separate table, you can potentially improve the performance of queries against the archive-enabled table.
- You can modify queries to include or exclude archive table data without having to change the SQL statement and prepare the application again. Instead, you can control the scope of the query with a global variable.
- You can store archive tables on a lower-cost device to reduce operating costs.

To create an archive table, follow the instructions in Creating an archive table (Db2 Administration Guide). When you create an archive table, use the SYSIBMADM.MOVE_TO_ARCHIVE global variable to specify whether Db2 is to automatically archive any rows that are deleted from the archive-enabled table.

When you query an archive-enabled table, you can specify whether you want the query to consider rows in the archive table. You do not have to modify the SQL. Instead, you can control the scope of the query by using the SYSIBMADM.GET_ARCHIVE global variable and the ARCHIVESENSITIVE bind option. To retrieve data from an archive able, set SYSIBMADM.GET_ARCHIVE to Y and bind the plan or package with ARCHIVESENSITIVE(YES).

The ARCHIVESENSITIVE bind option has no affect on the SYSIBMADM.MOVE_TO_ARCHIVE value.

You can later remove the relationship between the archive-enabled table and the archive table. To remove this relationship, issue the ALTER TABLE statement for the archive-enabled table and specify the DISABLE ARCHIVE clause. Both tables still exist, but the relationship is removed.

Related reference

Built-in global variables (Db2 SQL)

ARCHIVESENSITIVE bind option (Db2 Commands)

Creation of base tables

You use the CREATE TABLE statement to create a base table that you have designed.

When you create a table, Db2 records a definition of the table in the Db2 catalog. Creating a table does not store the application data. You can put data into the table by using several methods, such as the LOAD utility or the INSERT statement.

GUPI

Example: The following CREATE TABLE statement creates the EMP table, which is in a database named MYDB and in a table space named MYTS:

```
CREATE TABLE EMP
      (EMPNO     CHAR(6)         NOT NULL,
       FIRSTNME  VARCHAR(12)     NOT NULL,
       LASTNAME  VARCHAR(15)     NOT NULL,
       DEPT      CHAR(3)                 ,
       HIREDATE  DATE                    ,
       JOB       CHAR(8)                 ,
       EDL       SMALLINT                ,
       SALARY    DECIMAL(9,2)            ,
       COMM      DECIMAL(9,2)            ,
```

```
       PRIMARY KEY (EMPNO))
IN MYDB.MYTS;
```

The preceding CREATE TABLE statement shows the definition of multiple columns.

GUPI

Related concepts

Db2 columns

A column definition has two basic components, the column name and the data type.

Related reference

CREATE TABLE (Db2 SQL)

Creation of temporary tables

Temporary tables can help you identify a small subset of rows from an intermediate result table that you want to store permanently. The two types of temporary tables are created temporary tables and declared temporary tables.

You can use temporary tables to sort large volumes of data and to query that data. Then, when you have identified the smaller number of rows that you want to store permanently, you can store them in a base table. The two types of temporary tables in Db2 are the created temporary table and the declared temporary table. The following topics describe how to define each type.

Created temporary table

Sometimes you need a permanent, shareable description of a table but need to store data only for the life of an application process. In this case, you can define and use a created temporary table. Db2 does not log operations that it performs on created temporary tables; therefore, SQL statements that use them can execute more efficiently. Each application process has its own instance of the created temporary table.

Example: The following statement defines a created temporary table, TEMPPROD:

```
CREATE GLOBAL TEMPORARY TABLE TEMPPROD
  (SERIALNO     CHAR(8)     NOT NULL,
   DESCRIPTION  VARCHAR(60) NOT NULL,
   MFGCOSTAMT   DECIMAL(8,2)        ,
   MFGDEPTNO    CHAR(3)             ,
   MARKUPPCT    SMALLINT            ,
   SALESDEPTNO  CHAR(3)             ,
   CURDATE      DATE        NOT NULL);
```

Declared temporary table

Sometimes you need to store data for the life of an application process, but you do not need a permanent, shareable description of the table. In this case, you can define and use a declared temporary table.

Unlike other Db2 DECLARE statements, DECLARE GLOBAL TEMPORARY TABLE is an executable statement that you can embed in an application program or issue interactively. You can also dynamically prepare the statement.

When a program in an application process issues a DECLARE GLOBAL TEMPORARY TABLE statement, Db2 creates an empty instance of the table. You can populate the declared temporary table by using INSERT statements, modify the table by using searched or positioned UPDATE or DELETE statements, and query the table by using SELECT statements. You can also create indexes on the declared temporary table. The definition of the declared temporary table exists as long as the application process runs.

GUPI

Example: The following statement defines a declared temporary table, TEMP_EMP. (This example assumes that you have already created the WORKFILE database and corresponding table space for the temporary table.)

```
DECLARE GLOBAL TEMPORARY TABLE  SESSION.TEMP_EMP
   (EMPNO  CHAR(6)        NOT NULL,
```

```
    SALARY DECIMAL(9, 2)          ,
    COMM   DECIMAL(9, 2));
```

GUPI

If specified explicitly, the qualifier for the name of a declared temporary table, must be SESSION. If the qualifier is not specified, it is implicitly defined to be SESSION.

At the end of an application process that uses a declared temporary table, Db2 deletes the rows of the table and implicitly drops the description of the table.

Related concepts

Temporary tables (Db2 Application programming and SQL)

Related reference

CREATE GLOBAL TEMPORARY TABLE (Db2 SQL)

DECLARE GLOBAL TEMPORARY TABLE (Db2 SQL)

Creation of materialized query tables

Materialized query tables improve the performance of complex queries that operate on large amounts of data.

Using a materialized query table, Db2 pre-computes the results of data that is derived from one or more tables. When you submit a query, Db2 can use the results that are stored in a materialized query table rather than compute the results from the underlying source tables on which the materialized query table is defined. If the rewritten query is less costly, Db2 chooses to optimize the query by using the rewritten query, a process called *automatic query rewrite*.

To take advantage of automatic query rewrite, you must define, populate, and periodically refresh the materialized query table. You use the CREATE TABLE statement to create a table as a materialized query table.

GUPI

Example: The following CREATE TABLE statement defines a materialized query table named TRANSCNT. TRANSCNT summarizes the number of transactions in table TRANS by account, location, and year:

```
CREATE TABLE TRANSCNT (ACCTID, LOCID, YEAR, CNT) AS
  (SELECT ACCOUNTID, LOCATIONID, YEAR, COUNT(*)
     FROM TRANS
     GROUP BY ACCOUNTID, LOCATIONID, YEAR )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED
     MAINTAINED BY SYSTEM
     ENABLE QUERY OPTIMIZATION;
```

The fullselect, together with the DATA INITIALLY DEFERRED clause and the REFRESH DEFERRED clause, defines the table as a materialized query table.

GUPI

Related tasks

Dropping and re-creating a table space to change its attributes (Db2 Administration Guide)

Creation of a table with table-controlled partitioning

Table-controlled partitioning does not require an index for partitioning and is defined by PARTITION clauses on the CREATE TABLE statement.

When you define a partitioning index on a table in a partitioned table space, you specify the partitioning key and the limit key values in the PARTITION clause of the CREATE INDEX statement. This type of partitioning is known as *index-controlled partitioning*. Because the index is created separately from the associated table, you cannot insert data into the table until the partitioning index is created.

Db2 also supports a method called *table-controlled partitioning* for defining table partitions. You can use table-controlled partitioning instead of index-controlled partitioning.

With table-controlled partitioning, you identify column values that delimit partition boundaries with the PARTITION BY clause and the PARTITION ENDING AT clause of the CREATE TABLE statement. When you use this type of partitioning, an index is not required for partitioning.

GUPI

Example: Assume that you need to create a large transaction table that includes the date of the transaction in a column named POSTED. You want to keep the transactions for each month in a separate partition. To create the table, use the following statement:

```
CREATE TABLE TRANS
   (ACCTID ...,
    STATE ...,
    POSTED ...,
    ... , ...)
   PARTITION BY (POSTED)
   (PARTITION 1 ENDING AT ('01/31/2003'),
    PARTITION 2 ENDING AT ('02/28/2003'),
    ...
    PARTITION 13 ENDING AT ('01/31/2004'));
```

GUPI

Related concepts

Indexes on partitioned tables

The following types of indexes apply to only partitioned tables: partitioned indexes, partitioning indexes (PIs), data-partitioned secondary indexes (DPSIs), and nonpartitioned secondary indexes (NPIs or NPSIs).

Creation of temporal tables

A *temporal table* is a table that records the period of time when a row is valid.

You can create application-period temporal tables and system-period temporal tables. An *application-period temporal table* includes an application period, which is a period in which you maintain the beginning and ending values for a row. A *system-period temporal table* has a system period, and you can define system-period data versioning on the table to manage historical and current table data. A *bitemporal table* is a table that is both a system-period temporal table and an application-period temporal table.

System-period data versioning specifies that old rows are archived into another table. The table that contains the current active rows of a table is called the system-period temporal table. The table that contains the archived rows is called the history table. When you define a base table to use system-period data versioning or when you define system-period data versioning on an existing table, you must create a history table. You must specify a name for the history table and create a table space to hold that table.

When you update or delete a row in a system-period temporal table, Db2 inserts the previous version of the row into the history table. The historical versions of rows are written to the history table to record committed versions of the data in the associated system-period temporal table. Intermediate or uncommitted versions of rows in a system-period temporal table are not recorded in the history table. If a row in a system-period temporal table is updated multiple times within a single unit of work, and then a commit occurs, only one new historical version of that row is recorded in the history table. If a row is inserted into a system-period temporal table, and the insert is rolled back, nothing is recorded in the history table for the insert that was never committed. You can query a system-period temporal table with timestamp criteria to retrieve previous data values. You can specify the timestamp criteria in the query or by using special registers.

You can use system-period data versioning instead of developing your own programs for maintaining multiple versions of data within a database. With Db2, system-period data versioning is a more efficient method for maintaining versioned data.

You can use system-period temporal tables to track auditing information about when data changes occurred. If you want to track more information, such as who changed the data and the SQL operation that changed the data, you can include non-deterministic generated expression columns in the system-period temporal table.

Related concepts

Temporal tables and data versioning (Db2 Administration Guide)

Scenario for tracking auditing information (Db2 Administration Guide)

Related tasks

Querying temporal tables (Db2 Administration Guide)

Creation of table spaces

Often you do not need to create a table space before you create a table. You only need to create a table space explicitly when you define a declared temporary table or if you manage all of your own data sets.

Universal table spaces

Universal table spaces (UTS) combine the benefits of data partitions and segmented organization. Each UTS table space always contains only a single table.

Universal table spaces offer the following advantages, when compared to the deprecated non-UTS table space types:

- A choice of partitioning methods:
  - Partitions based on ranges of data values, with Partition-by-range table spaces (Db2 Administration Guide)
  - Partitions based on data growth and automatically managed by Db2, with Partition-by-growth table spaces (Db2 Administration Guide)
- Improved space management for varying-length rows because a segmented space-map page has more information about free space than a partitioned space-map page
- Improved mass delete performance because mass delete in a segmented table space organization tends to be faster than in non-segmented table space organizations
- Table scans that are localized to segments
- Immediate reuse of all or most of the segments of a table after the table is dropped or mass deleted

Comparison of table space types

The following table compares the characteristics of the various table space types that Db2 for z/OS supports.

Table 11. Comparison of table space types

Type	Segmented?	Partitioned?	Remarks
Partition-by-range table space	Yes	Yes, based on data value ranges	
Partition-by-growth table space	Yes	Yes, based on data growth	
LOB table space	No	No	For auxiliary tables that contain the data for LOB columns.
XML table space	Yes	Yes, based on the partitioning method of the base table.	UTS table spaces that are created implicitly for XML data.
Partitioned (non-UTS) table space	No	Yes, based on data value ranges	This type is deprecated.
Segmented (non-UTS) table space	Yes	No	This type is deprecated.

Table 11. Comparison of table space types (continued)

Type	Segmented?	Partitioned?	Remarks
Simple table space	No	No	This type is deprecated."1" on page 97

Notes:

1. Db2 11 does not support creating simple table spaces. Existing simple table spaces remain supported, but they are likely to be unsupported in the future.

Implicit creation of table spaces

Db2 creates a table space implicitly if you issue a CREATE TABLE statement without specifying an existing table space name. Db2 derives the table space name from the name of your table. The table space is created in the either the specified database or, if the database is not specified, in an implicitly created database .

If the table contains a LOB column and SQLRULES are STD, Db2 also creates the LOB table space, the auxiliary table, and an auxiliary index. Db2 also creates all underlying XML objects. In this case, Db2 uses the default storage group, SYSDEFLT.

If you create a table space implicitly, Db2 uses defaults for the space allocation attributes. The default values of PRIQTY and SECQTY specify the space allocation for the table space. If the value of the TSQTY subsystem parameter is nonzero, it determines the default values for PRIQTY and SECQTY. If the value of TSQTY is zero, the default values for PRIQTY and SECQTY are determined as described in the CREATE TABLESPACE statement.

Explicit creation of table spaces

If you must create table spaces explicitly, you can issue CREATE TABLESPACE statements. It is best to create universal partition-by-growth or partition-by-range table spaces. Non-UTS table spaces for base tables are deprecated and likely to be unsupported in the future.

When you create or alter a table space, the clauses that you specify in a CREATE TABLESPACE or ALTER TABLESPACE statement determine the type of table space.

The following table shows the result.

Table 12. Table space types and related clauses

Table space type to create	Clauses to specify
Partition-by-growth	Any of the following combinations: • MAXPARTITIONS and NUMPARTS • MAXPARTITIONS and SEGSIZE *n*"1" on page 98 • MAXPARTITIONS
Partition-by-range	NUMPARTS and SEGSIZE *n*"1" on page 98
Segmented (non-UTS) "2" on page 98	One of the following combinations: • SEGSIZE *n*"1" on page 98 • Omit MAXPARTITIONS, NUMPARTS, and SEGSIZE
Partitioned (non-UTS) "2" on page 98	NUMPARTS and SEGSIZE 0

Table 12. Table space types and related clauses (continued)

Table space type to create	Clauses to specify

Notes:

1. Where *n* is a non-zero value. The DPSEGSZ subsystem parameter determines the default value. For more information, see DEFAULT PARTITION SEGSIZE field (DPSEGSZ subsystem parameter) (Db2 Installation and Migration).
2. Non-UTS table spaces for base tables are deprecated and likely to be unsupported in the future.

Related concepts

Db2 table spaces

A Db2 *table space* is a set of volumes on disks that hold the data sets in which tables are actually stored. Every table is stored in table space.

Implicitly defined table spaces (Db2 Administration Guide)

Related tasks

Creating table spaces explicitly (Db2 Administration Guide)

Creating partition-by-range table spaces (Db2 Administration Guide)

Creating partition-by-growth table spaces (Db2 Administration Guide)

Related reference

CREATE TABLESPACE (Db2 SQL)

Assignment of table spaces to physical storage

You can store table spaces and index spaces in user-managed storage, SMS-managed storage, or in Db2-managed storage groups. (A *storage group* is a set of disk volumes.)

If you do not use SMS, you need to name the Db2 storage groups when you create table spaces or index spaces. Db2 allocates space for these objects from the named storage group. You can assign different partitions of the same table space to different storage groups.

Recommendation: Use products in the IBM Storage Management Subsystem (SMS) family, such as Data Facility SMS (DFSMS), to manage some or all of your data sets. Organizations that use SMS to manage Db2 data sets can define storage groups with the VOLUMES(*) clause. You can also assign management class, data class, and storage class attributes. As a result, SMS assigns a volume to the table spaces and index spaces in that storage group.

The following figure shows how storage groups work together with the various Db2 data structures.

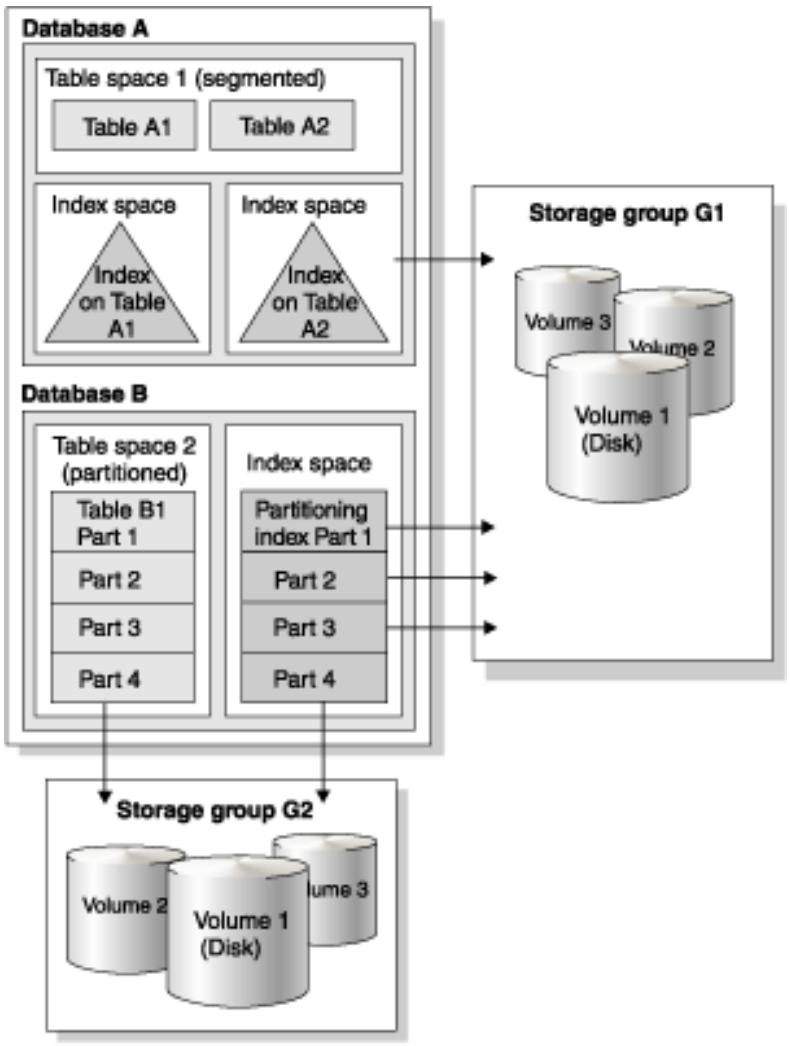


Figure 15. Hierarchy of Db2 structures

To create a Db2 storage group, use the SQL statement CREATE STOGROUP. Use the VOLUMES(*) clause to specify the SMS management class (MGMTCLAS), SMS data class (DATACLAS), and SMS storage class (STORCLAS) for the Db2 storage group.

After you define a storage group, Db2 stores information about it in the Db2 catalog. The catalog table SYSIBM.SYSSTOGROUP has a row for each storage group, and SYSIBM.SYSVOLUMES has a row for each volume in the group.

The process of installing Db2 includes the definition of a default storage group, SYSDEFLT. If you have authorization, you can define tables, indexes, table spaces, and databases. Db2 uses SYSDEFLT to allocate the necessary auxiliary storage. Db2 stores information about SYSDEFLT and all other storage groups in the catalog tables SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES.

Recommendation: Use storage groups whenever you can, either explicitly or implicitly (by using the default storage group). In some cases, organizations need to maintain closer control over the physical storage of tables and indexes. These organizations choose to manage their own user-defined data sets rather than using storage groups. Because this process is complex, this information does not describe the details.

GUPI

Example: Consider the following CREATE STOGROUP statement:

```
CREATE STOGROUP MYSTOGRP
  VOLUMES (*)
  VCAT ALIASICF;
```

This statement creates storage group MYSTOGRP. The asterisk (*) on the VOLUMES clause indicates that SMS is to manage your storage group. The VCAT *catalog-name* clause identifies ALIASICF as the name or alias of the catalog of the integrated catalog facility that the storage group is to use. The catalog of the integrated catalog facility stores entries for all data sets that Db2 creates on behalf of a storage group.

The data sets are linear VSAM data sets cataloged in the integrated catalog facility catalog that *catalog-name* identifies. For more information about *catalog-name* values, see Naming conventions (Db2 SQL).

GUPI

IBM Storage Management Subsystem

Db2 for z/OS includes the Storage Management Subsystem (SMS) capabilities. A key product in the SMS family is the Data Facility Storage Management Subsystem (DFSMS). DFSMS can automatically manage all the data sets that Db2 uses and requires. If you use DFSMS to manage your data sets, the result is a reduced workload for Db2 database administrators and storage administrators.

You can experience the following benefits by using DFSMS:

- Simplified data set allocation
- Improved allocation control
- Improved performance management
- Automated disk space management
- Improved management of data availability
- Simplified data movement

Db2 database administrators can use DFSMS to achieve all their objectives for data set placement and design. To successfully use DFSMS, Db2 database administrators and storage administrators need to work together to ensure that the needs of both groups are satisfied.

Related concepts

Db2 storage groups

Db2 *storage groups* are a set of volumes on disks that hold the data sets in which tables and indexes are stored.

Related tasks

Choosing data page sizes (Db2 Performance)

Related reference

CREATE STOGROUP (Db2 SQL)

Creation of indexes

Indexes provide efficient access to table data, but can require additional processing when you modify data in a table.

You can create indexes to enforce uniqueness. You can also choose to use indexes because of access requirements.

Using indexes involves a trade-off. A greater number of indexes can simultaneously improve the performance of a certain transaction and require additional processing for inserting, updating, and deleting index keys.

After you create an index, Db2 maintains the index, but you can perform necessary maintenance, such as reorganizing it or recovering it, as necessary.

Related concepts

Indexes on table columns (Db2 Administration Guide)

Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX') (Db2 Performance)

Related tasks

Designing indexes for performance (Db2 Performance)

Related information

Implementing Db2 indexes (Db2 Administration Guide)

Types of indexes

In Db2 for z/OS, you can create a number of different types of indexes. Carefully consider which type or types best suit your data and applications.

All of the index types are listed in the following tables. These index types are not necessarily mutually exclusive. For example, a unique index can also be a clustering index. Restrictions are noted.

The following table lists the types of indexes that you can create on any table.

Table 13. Index types that are applicable to any table type

Type	Description	SQL syntax	More information
Unique index	An index that ensures that the value in a particular column or set of columns is unique.	`CREATE INDEX... UNIQUE...`	"Unique indexes" on page 106 "Nonunique indexes" on page 108
Primary index	A unique index on the primary key of the table. A *primary key* is column or set of columns that uniquely identifies one row of a table. You define a primary key when you create or alter a table; specify PRIMARY KEY in the CREATE TABLE statement or ALTER TABLE statement. Primary keys are optional. If you define a primary key on a table, you must define a primary index on that key. Otherwise, if the table does not have a primary key, it cannot have a primary index. Each table can have only one primary index. However, the table can have additional unique indexes.	No keywords are required in the CREATE INDEX statement. An index is a primary index if the index key that is specified in the CREATE INDEX statement matches the primary key of the table.	Defining a parent key and unique index (Db2 Application programming and SQL)
Secondary index	An index that is not a primary index. In the context of a partitioned table, a secondary index can also mean an index that is not a partitioning index. See Table 14 on page 102.	No keywords are required in the CREATE INDEX statement. A secondary index is any index that is not a primary index or partitioning index.	None

Clustering index	An index that ensures a logical grouping. When data is inserted into the table, the clustering index attempts to maintain the clustering sequence within the partition.
	Each table can have only one clustering index.
		`CREATE INDEX... CLUSTER...`
		or
		`ALTER INDEX... CLUSTER...`
			"Clustering indexes" on page 108

Table 13. Index types that are applicable to any table type (continued)

Type	Description	SQL syntax	More information
Expression-based index	An index that is based on a general expression. Use expression-based indexes when you want an efficient evaluation of queries that involve a column-expression.	In the CREATE INDEX or ALTER INDEX statement, the index key is defined as an expression rather than a column or set of columns.	"Expression-based indexes" on page 110

The following table lists the types of indexes that you can create on partitioned tables. These indexes apply to partition-by-range table spaces. They do not apply to partition-by-growth table spaces.

Table 14. Index types that are applicable to only partitioned tables

Type	Description	SQL syntax	More information
Partitioned index	An index that is physically partitioned. A partitioned index consists of multiple data sets. Each data set corresponds to a table partition.	`CREATE INDEX... PARTITIONED...`	"Indexes on partitioned tables" on page 112
Partitioning index (PI)	An index that corresponds to the columns that partition the table. These columns are called the *partitioning key* and are specified in the PARTITION BY clause of the CREATE TABLE statement. All partitioning indexes must also be partitioned. Partitioning indexes are not required.	No keywords are required in the CREATE INDEX statement. An index is a partitioning index if the index key matches the partitioning key. To confirm that an index is partitioning index, check the SYSIBM.SYSINDEXES catalog table. The INDEXTYPE column for that index contains a P if the index is a partitioning index.	"Indexes on partitioned tables" on page 112
Secondary index	Depending on the context, a secondary index can mean one of the following two things: • An index that is not a partitioning index. • An index that is not a primary index.	No keywords are required in the CREATE INDEX statement. A secondary index is any index that is not a primary index or partitioning index.	"Indexes on partitioned tables" on page 112
Data partitioned secondary index (DPSI)	A partitioned index that is not a partitioning index. These indexes are also called partitioned secondary indexes (PSIs).	`CREATE INDEX... PARTITIONED...` Also, the specified index key must not match the partitioning key.	"Indexes on partitioned tables" on page 112
Nonpartitioned secondary index (NPSI)	An index that is not partitioned or partitioning. These indexes are also called nonpartitioned indexes (NPIs).	The CREATE INDEX statement does not include the PARTITIONED keyword. Also, the index key does not match the partitioning key.	"Indexes on partitioned tables" on page 112

Table 14. Index types that are applicable to only partitioned tables (continued)

Type	Description	SQL syntax	More information

Multi-piece index	A nonpartitioned index that has multiple data sets. The data sets do not correspond to data partitions.
	
	Use a multi-piece index to spread a large index across multiple data sets and thus reduce the physical I/O contention on the index.
		`CREATE INDEX... PIECESIZE ...`
		
		or
		
		`ALTER INDEX... PIECESIZE ...`
			None

The following table lists the XML index type.

Table 15. Index types that are applicable to only tables with XML columns

Type	Description	SQL syntax	More information

XML index	An index that uses a particular XML pattern expression to index paths and values in XML documents that are stored in a single XML column.	`CREATE INDEX... GENERATE KEY USING XMLPATTERN`
		
		or
		
		`CREATE INDEX... GENERATE KEYS USING XMLPATTERN`
			"XML index attributes" on page 111

Additionally, when you create any of these types of indexes, you can define whether they have the following characteristics:

Table 16. General index characteristics

Characteristic	Description	SQL syntax	More information

Padded	Any varying-length string columns in the index are padded with the default pad character to their maximum length.	`CREATE INDEX... PADDED ...`
		
		or
		
		`ALTER INDEX... PADDED...`
			"Indexes that are padded or not padded" on page 110

Compressed	The data is compressed to reduce the size of the index on disk.	`CREATE INDEX... COMPRESS YES...`
		
		or
		
		`ALTER INDEX... COMPRESS YES...`
			"Compression of indexes" on page 111

Related concepts

Index keys

The usefulness of an index depends on the design of its key, which you define at the time that you create the index.

Related reference

ALTER INDEX (Db2 SQL)

CREATE INDEX (Db2 SQL)

Related information

Implementing Db2 indexes (Db2 Administration Guide)

How indexes can help to avoid sorts

Db2 can use indexes to avoid sorts when processing queries with the ORDER BY clause.

When a query contains an ORDER BY clause, Db2 looks for indexes that satisfy the order in the query. For Db2 to be able to use an index to access ordered data, you must define an index on the same columns as specified in the ORDER BY clause.

Forward index scan

For Db2 to use a forward index scan, the ordering must be exactly the same as in the ORDER BY clause.

Backward index scan

For Db2 to use a backward index scan, the ordering must be exactly the opposite of what is requested in the ORDER BY clause.

GUPI

Example 1: For example, if you define an index by specifying DATE DESC, TIME ASC as the column names and order, Db2 can use this same index for both of the following ORDER BY clauses:

- Forward scan for ORDER BY DATE DESC, TIME ASC
- Backward scan for ORDER BY DATE ASC, TIME DESC

You do not need to create two indexes for the two ORDER BY clauses. Db2 can use the same index for both forward index scan and backward index scan.

GUPI

In addition to forward and backward scans, you have the option to create indexes with a pseudo-random order. This ordering option is useful when ascending insertions or hotspots cause contention within the indexes. Indexes created with the RANDOM option do not support range scans. They do support equality lookups.

GUPI

Example 2: Suppose that the query includes a WHERE clause with a predicate of the form COL=*constant*. For example:

```
...
WHERE CODE = 'A'
ORDER BY CODE, DATE DESC, TIME ASC
```

Db2 can use any of the following index keys to satisfy the ordering:

- CODE, DATE DESC, TIME ASC
- CODE, DATE ASC, TIME DESC
- DATE DESC, TIME ASC
- DATE ASC, TIME DESC

Db2 can ignore the CODE column in the ORDER BY clause and the index because the value of the CODE column in the result table of the query has no effect on the order of the data. If the CODE column is included, it can be in any position in the ORDER BY clause and in the index.

GUPI

Index keys

The usefulness of an index depends on the design of its key, which you define at the time that you create the index.

An *index key* is a column, an ordered collection of columns, or an expression on which you define an index. Db2 uses an index key to determine the order of index entries. Good candidates for index keys are columns or expressions that you use frequently in operations that select, join, group, and order data.

All index keys do not need to be unique. For example, an index on the SALARY column of the sample EMP table allows duplicates because several employees can earn the same salary.

A *composite key* is an index key that is built on 2 or more columns. An index key can contain up to 64 columns.

GUPI

For example, the following SQL statement creates a unique index on the EMPPROJACT table. A composite key is defined on two columns, PROJNO and STDATE.

```
CREATE UNIQUE INDEX XPROJAC1
  ON EMPPROJACT
     (PROJNO ASC,
          STDATE ASC)
⋮
```

This composite key is useful when you need to find project information by start date. Consider a SELECT statement that has the following WHERE clause:

```
WHERE PROJNO='MA2100' AND STDATE='2004-01-01'
```

This SELECT statement can execute more efficiently than if separate indexes are defined on PROJNO and on STDATE.

GUPI

In general, try to create an index that is selective, because the more selective an index is, the more efficient it is. An efficient index contains multiple columns, is ordered in the same sequence as the SQL statement, and is used often in SQL statements. To create an efficient index, consider the following recommendations when you create an index and define the index keys:

- Define as few indexes as possible on a column that is updated frequently because every change to the column data must be reflected in each index.
- Consider using a composite key, which might be more useful than a key on a single column when the comparison is for equality. A single multicolumn index is more efficient when the comparison is for equality and the initial columns are available. However, for more general comparisons, such as A > *value* AND B > *value*, multiple indexes might be more efficient.

Related concepts

Query and application performance analysis
Performance analysis for queries and applications begins by answering some basic questions.

General index attributes

You typically determine which type of index you need to define after you define a table space. An index can have many different attributes.

Index attributes fall into two broad categories: general attributes that apply to indexes on all tables and specific attributes that apply to indexes on partitioned tables only. The following table summarizes these categories.

Table 17. Index attributes

Table or table space type	Index attribute
Any	• Unique or nonunique • Clustering or nonclustering • Padded or not padded • Exclude nulls
Partitioned	• Partitioning • Secondary

This topic explains the types of indexes that apply to all tables. Indexes that apply to partitioned tables only are covered separately.

Related concepts

Indexes on partitioned tables

The following types of indexes apply to only partitioned tables: partitioned indexes, partitioning indexes (PIs), data-partitioned secondary indexes (DPSIs), and nonpartitioned secondary indexes (NPIs or NPSIs).

Unique indexes

Db2 uses unique indexes to ensure that no identical key values are stored in a table.

When you create a table that contains a primary key or a unique constraint, you must create a unique index for the primary key and for each unique constraint. Db2 marks the table definition as incomplete until the explicit creation of the required enforcing indexes, which can be created implicitly depending on whether the table space was created implicitly, the schema processor, or the CURRENT RULES special register. If the required indexes are created implicitly, the table definition is not marked as incomplete.

Restrict access with unique indexes

You can also use indexes to meet access requirements.

Example 1: A good candidate for a unique index is the EMPNO column of the EMP table. The following figure shows a small set of rows from the EMP table and illustrates the unique index on EMPNO.

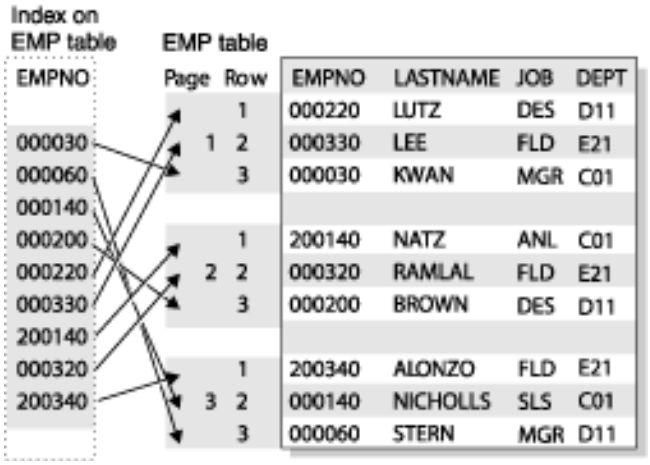


Figure 16. A unique index on the EMPNO column

Db2 uses this index to prevent the insertion of a row to the EMP table if its EMPNO value matches that of an existing row. The preceding figure illustrates the relationship between each EMPNO value in the index

and the corresponding page number and row. Db2 uses the index to locate the row for employee 000030, for example, in row 3 of page 1.

If you do not want duplicate values in the key column, create a unique index by using the UNIQUE clause of the CREATE INDEX statement.

GUPI

Example 2: The DEPT table does not allow duplicate department IDs. Creating a unique index, as the following example shows, prevents duplicate values.

```
CREATE UNIQUE INDEX MYINDEX
   ON DEPT (DEPTNO);
```

The index name is MYINDEX, and the indexed column is DEPTNO.

If a table has a primary key (as the DEPT table has), its entries must be unique. Db2 enforces this uniqueness by defining a unique index on the primary key columns, with the index columns in the same order as the primary key columns.

GUPI

Before you create a unique index on a table that already contains data, ensure that no pair of rows has the same key value. If Db2 finds a duplicate value in a set of key columns for a unique index, Db2 issues an error message and does not create the index.

If an index key allows nulls for some of its column values, you can use the WHERE NOT NULL clause to ensure that the non-null values of the index key are unique.

Unique indexes are an important part of implementing referential constraints among the tables in your Db2 database. You cannot define a foreign key unless the corresponding primary key already exists and has a unique index defined on it.

When not to use a unique index

In some cases you might not want to use a unique index. You can improve the performance of data access when the values of the columns in the index are not necessarily unique by creating a default index.

When you create a default index, Db2 allows you to enter duplicate values in a key column.

For example, assume that more than one employee is named David Brown. Consider an index that is defined on the FIRSTNME and LASTNAME columns of the EMP table.

GUPI

```
CREATE INDEX EMPNAME ON EMP (FIRSTNME, LASTNAME);
```

GUPI

This is an example of an index that can contain duplicate entries.

Tip: Do not create this type of index on very small tables because scans of the tables are more efficient than using indexes.

INCLUDE columns

Unique indexes can include additional columns that are not part of a unique constraint. Those columns are called INCLUDE columns. When you specify INCLUDE columns in a unique index, queries can use the unique index for index-only access. Including these columns can eliminate the need to maintain extra indexes that are used solely to enable index-only access.

Related reference

CREATE INDEX (Db2 SQL)

Nonunique indexes

You can use nonunique indexes to improve the performance of data access when the values of the columns in the index are not necessarily unique.

Recommendation: Do not create nonunique indexes on very small tables, because scans of the tables are more efficient than using indexes.

To create nonunique indexes, use the SQL CREATE INDEX statement. For nonunique indexes, Db2 allows users and programs to enter duplicate values in a key column.

GUPI

Example: Assume that more than one employee is named David Brown. Consider an index that is defined on the FIRSTNME and LASTNAME columns of the EMP table.

```
CREATE INDEX EMPNAME
   ON EMP (FIRSTNME, LASTNAME);
```

This index is an example of a nonunique index that can contain duplicate entries.

GUPI

Related tasks

Designing indexes for performance (Db2 Performance)

Related reference

CREATE INDEX (Db2 SQL)

Clustering indexes

A *clustering index* determines how rows are physically ordered (clustered) in a table space. Clustering indexes provide significant performance advantages in some operations, particularly those that involve many records. Examples of operations that benefit from clustering indexes include grouping operations, ordering operations, and comparisons other than equal.

Any index, except for an expression-based index or an XML index, can be a clustering index. You can define only one clustering index on a table.

You can define a clustering index on a partitioned table space or on a segmented table space. On a partitioned table space, a clustering index can be a partitioning index or a secondary index. If a clustering index on a partitioned table is not a partitioning index, the rows are ordered in cluster sequence within each data partition instead of spanning partitions. (Prior to Version 8 of Db2 UDB for z/OS, the partitioning index was required to be the clustering index.)

Restriction: An expression based index or an XML index cannot be a clustering index.

When a table has a clustering index, an INSERT statement causes Db2 to insert the records as nearly as possible in the order of their index values. The first index that you define on the table serves implicitly as the clustering index unless you explicitly specify CLUSTER when you create or alter another index. For example, if you first define a unique index on the EMPNO column of the EMP table, Db2 inserts rows into the EMP table in the order of the employee identification number unless you explicitly define another index to be the clustering index.

Although a table can have several indexes, only one index can be a clustering index. If you do not define a clustering index for a table, Db2 recognizes the first index that is created on the table as the implicit clustering index when it orders data rows.

Tip:

- Always define a clustering index. Otherwise, Db2 might not choose the key that you would prefer for the index.
- Define the sequence of a clustering index to support high-volume processing of data.

You use the CLUSTER clause of the CREATE INDEX or ALTER INDEX statement to define a clustering index.

GUPI

Example: Assume that you often need to gather employee information by department. In the EMP table, you can create a clustering index on the DEPTNO column.

```
CREATE INDEX DEPT_IX
  ON EMP
    (DEPTNO ASC)
     CLUSTER;
```

As a result, all rows for the same department are probably close together. Db2 can generally access all the rows for that department in a single read. (Using a clustering index does not guarantee that all rows for the same department are stored on the same page. The actual storage of rows depends on the size of the rows, the number of rows, and the amount of available free space. Likewise, some pages may contain rows for more than one department.)

GUPI

The following figure shows a clustering index on the DEPT column of the EMP table; only a subset of the rows is shown.

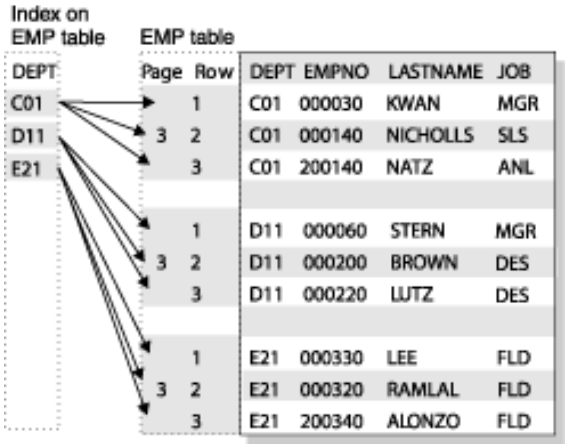


Figure 17. A clustering index on the EMP table

Suppose that you subsequently create a clustering index on the same table. In this case, Db2 identifies it as the clustering index but does not rearrange the data that is already in the table. The organization of the data remains as it was with the original nonclustering index that you created. However, when the REORG utility reorganizes the table space, Db2 clusters the data according to the sequence of the new clustering index. Therefore, if you know that you want a clustering index, you should define the clustering index before you load the table. If that is not possible, you must define the index and then reorganize the table. If you create or drop and re-create a clustering index after loading the table, those changes take effect after a subsequent reorganization.

Related reference

Employee table (DSN8B10.EMP)
The sample employee table identifies all employees by an employee number and lists basic personnel information.

CREATE INDEX (Db2 SQL)

Indexes that exclude NULL keys

You can exclude NULL keys from an index to reduce the size of an index and improve the performance of an index.

Some table values are never used in queries and are unnecessary in an index. NULL key columns add to index size and can reduce the performance of index scans. If you exclude NULL key columns from an

index, Db2 only creates index entries for key columns that are not null. You can specify that an index excludes null keys when you create an index with the CREATE INDEX statement.

A NULL key in an index is not the same as a null foreign key.

Related reference
CREATE INDEX (Db2 SQL)

Indexes that are padded or not padded

The NOT PADDED and PADDED options of the CREATE INDEX and ALTER INDEX statements specify how varying-length string columns are stored in an index.

You can choose not to pad varying-length string columns in the index to their maximum length (the default), or you can choose to pad them.

GUPI If you specify the NOT PADDED clause on a CREATE INDEX statement, any varying-length columns in the index key are not padded to their maximum length. If an existing index key includes varying-length columns, you can consider altering the index to use the NOT PADDED clause. However, using the NOT PADDED clause on the ALTER INDEX statement to change the padding places the index in the REBUILD-pending (RBDP) state. You should rebuild the index to remove the RBDP state. GUPI

Using the NOT PADDED clause has the following advantages:

- Db2 can use index-only access for the varying-length columns within the index key, which enhances performance.
- Db2 stores only actual data, which reduces the storage requirements for the index key.

However, using the NOT PADDED clause might also have the following disadvantages:

- Index key comparisons are slower because Db2 must compare each pair of corresponding varying-length columns individually instead of comparing the entire key when the columns are padded to their maximum length.
- Db2 stores an additional 2-byte length field for each varying-length column. Therefore, if the length of the padding (to the maximum length) is less than or equal to 2 bytes, the storage requirements could actually be greater for varying-length columns that are not padded.

Tip: Use the NOT PADDED clause to implement index-only access if your application typically accesses varying-length columns.

To control whether varying length columns are padded by default, use the PAD INDEXES BY DEFAULT option on installation panel DSNTIPE.

Related reference
CREATE INDEX (Db2 SQL)

Expression-based indexes

By using the expression-based index capability of Db2, you can create an index that is based on a general expression. You can enhance query performance if Db2 chooses the expression-based index.

Use expression-based indexes when you want an efficient evaluation of queries that involve a column-expression. In contrast to simple indexes, where index keys consist of a concatenation of one or more table columns that you specify, the index key values are not the same as values in the table columns. The values have been transformed by the expressions that you specify.

You can create the index by using the CREATE INDEX statement, and specifying an expression, rather than a column name. If an index is created with the UNIQUE option, the uniqueness is enforced against the values that are stored in the index, not against the original column values.

Db2 does not use expression-based indexes for queries that use sensitive static scrollable cursors.

Related concepts
Expressions (Db2 SQL)
Index keys

The usefulness of an index depends on the design of its key, which you define at the time that you create the index.

Related reference
CREATE INDEX (Db2 SQL)

Compression of indexes
You can reduce the amount of space that an index occupies on disk by compressing the index.

The COMPRESS YES/NO clause of the ALTER INDEX and CREATE INDEX statements allows you to compress the data in an index and reduce the size of the index on disk. However, index compression is heavily data-dependent, and some indexes might contain data that does not yield significant space savings. Compressed indexes might also use more real and virtual storage than non-compressed indexes. The amount of additional real and virtual storage that is required depends on the compression ratio that is used for the compressed keys, the amount of free space, and the amount of space that is used by the key map.

You can choose 8 KB, 16 KB, and 32 KB buffer pool page sizes for the index. Use the DSN1COMP utility on existing indexes to estimate the appropriate page size for new indexes. Choosing a 32 KB buffer pool instead of a 16 KB or an 8 KB buffer pool accommodates a potentially higher compression ratio, but this choice also increases the potential to use more storage. Estimates for index space savings from the DSN1COMP utility, either on the true index data or some similar index data, are not exact.

If I/O is needed to read an index, the CPU degradation for a index scan is probably relatively small, but the CPU degradation for random access is likely to be very significant.

CPU degradation for deletes and updates is significant even if no read I/O is necessary.

Related reference
ALTER INDEX (Db2 SQL)
CREATE INDEX (Db2 SQL)

XML index attributes
You can create an index on an XML column for efficient evaluation of XQuery expressions to improve performance for queries on XML documents.

In simple relational indexes, index keys are composed of one or more table columns that you specified. However, an *XML index* uses a particular XML pattern expression to index paths and values in XML documents that are stored in a single XML column.

In an XML index, only the attribute nodes, text nodes, or element nodes that match the XML pattern expression are indexed. An XML index only indexes the nodes that match the specific XML pattern and not the document itself. Two more key fields are added to the index to form the composite index key. The extra key fields, which identify the XML document and the node position within the document, are stored in the catalog. These fields are not involved in uniqueness checking for unique indexes.

Use the CREATE INDEX statement with the XMLPATTERN keyword to create an XML index. You must also specify the XML path to be indexed. An index key is formed by concatenating the values that are extracted from the nodes in the XML document that satisfy the specified XML path with the document and node ID.

You specify a data type for every XML index. XML indexes support the following data types:

- VARCHAR
- DECFLOAT
- TIMESTAMP(12)
- DATE

You can use the following clauses to control whether Db2 inserts values into a table that are not compatible with the index data type:

- IGNORE INVALID VALUES
- REJECT INVALID VALUES

When you index an XML column with XMLPATTERN, only the parts of the document that satisfy the XML pattern expression are indexed. Multiple parts of the document might satisfy the XML pattern that you specified in the XMLPATTERN. Therefore, more than one index key entry might be generated and inserted into the index for the insertion of a single document.

Only one XML index specification is allowed per CREATE INDEX statement. However, you can create an XML index with multiple keys, or create multiple XML indexes on an XML column.

Restriction: Partitioned XML indexes are not supported

GUPI

Example 1: Assume that you must search for a specific employee's surname (name/last) on the employee elements. You can use the following CREATE INDEX statement to create an index on the `'/department/emp/name/last'` XML pattern expression:

```
CREATE INDEX EMPINDEX ON DEPARTMENT (DEPTDOCS)
        GENERATE KEYS USING XMLPATTERN '/department/emp/name/last'
        AS SQL VARCHAR(20)
```

After the EMPINDEX index is created successfully, several entries are populated in the catalog tables.

GUPI

Example 2: You can create two XML indexes with the same pattern expression by using different data types for each. You can use the different indexes to choose how you want to interpret the result of the expression as multiple data types. For example, the value '12345' has a character representation but it can also be interpreted as the number 12,345. For example, assume that you want to index the path `'/department/emp/@id'` as both a character string and a number. You must create two indexes, one for the VARCHAR data type and one for the DECFLOAT data type. The values in the document are cast to the specified data type for each index.

Related concepts

Storage structure for XML data (Db2 Programming for XML)

Processing XML data with Db2 pureXML

You can use pureXML with your client applications to manage XML data in Db2 tables. You can store well-formed XML documents in their hierarchical form and retrieve all or portions of those documents.

XML data indexing (Db2 Programming for XML)

Pattern expressions (Db2 Programming for XML)

Best practices for XML performance in Db2 (Db2 Performance)

Related reference

XMLEXISTS predicate (Db2 SQL)

CREATE INDEX (Db2 SQL)

Indexes on partitioned tables

The following types of indexes apply to only partitioned tables: partitioned indexes, partitioning indexes (PIs), data-partitioned secondary indexes (DPSIs), and nonpartitioned secondary indexes (NPIs or NPSIs).

Partitioned index

A *partitioned index* is an index that is physically partitioned. Any index on a partitioned table, except for an XML index, can be physically partitioned.

To create a partitioned index, specify PARTITIONED in the CREATE INDEX statement.

A partitioned index consists of multiple data sets. Each data set corresponds to a table partition. The following figure illustrates the difference between a partitioned index and a nonpartitioned index.

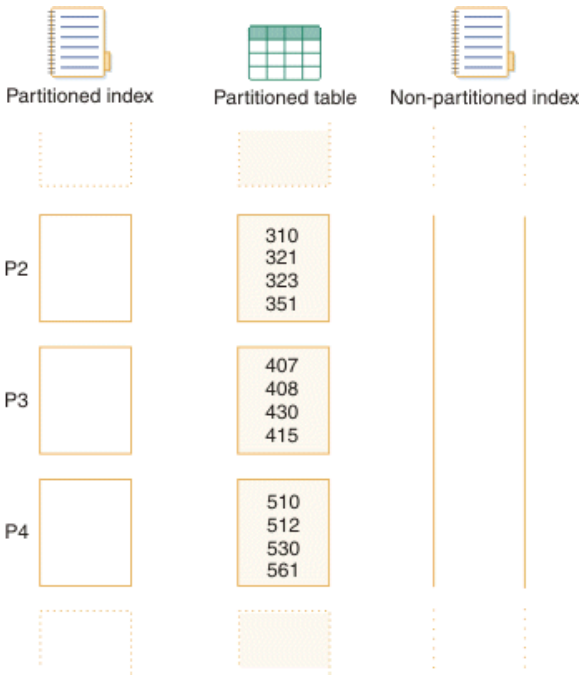


Figure 18. Comparison of partitioned and nonpartitioned index

Partitioning index

A *partitioning index* is an index on the column or columns that partition the table.

The CREATE INDEX statement does not have a specific SQL keyword that designates an index as a partitioning index. Instead, an index is a partitioning index if the index key that is specified in the CREATE INDEX statement matches the partitioning key. The *partitioning key* is the column or columns that are specified in the PARTITION BY clause of the CREATE TABLE statement. Those columns partition the table. An index key matches the partitioning key if it has the same leftmost columns and collating sequence (ASC/DESC) as the columns in the partitioning key.

A partitioning key is different from the limit key values. A partitioning key defines the columns on which the table is partitioned. The limit key values define which values belong in each partition. Specifically, a *limit key value* is the value of the partitioning key that defines the partition boundary. It is the highest value of the partitioning key for an ascending index, or the lowest value for a descending index. Limit key values are specified in the PARTITION... ENDING AT clause of a CREATE TABLE statement or ALTER TABLE statement. The specified ranges partition the table space and the corresponding partitioning index space.

Partitioning is different from clustering. *Clustering* is a grouping that controls how rows are physically ordered in a partition or table space. Clustering is controlled by a clustering index and can apply to any type of table space. Partitioning applies to only partitioned table spaces. *Partitioning* is a grouping that guarantees that rows are inserted into certain partitions based on certain value ranges as defined by the limit key values. You can however use clustering and partitioning together. If you use the same index for clustering and partitioning the rows are physically ordered across the entire table space.

Partitioning indexes are not the same as partitioned indexes. However, it is best that all partitioning indexes also be partitioned.

Partitioning indexes are not required, because the partitioning scheme (the partitioning key and limit key values) are already defined in the table definition.

You might work with partitioned tables that were created in older versions of Db2 and the partitioning scheme was not defined as part of the table definition. In this case, a partitioning index is required to specify the partitioning scheme. (The partitioning key and the limit key values were specified in the PART VALUES clause of the CREATE INDEX statement.) This behavior is called *index-controlled partitioning*. Db2 can still process those indexes and the partitioned tables. However, all new partitioned tables should

define the partitioning scheme in the table definition. This behavior is sometimes called *table-controlled partitioning* to distinguish it from the older index-controlled partitioning.

Restriction: You cannot create a partitioning index in a partition-by-growth table space.

Example: GUPI Assume that a table contains state area codes, and you need to create a partitioning index to sequence the area codes across partitions. You can use the following SQL statements to create the table and the partitioning index:

```
CREATE TABLE AREA_CODES
  (AREACODE_NO  INTEGER  NOT NULL,
   STATE        CHAR (2) NOT NULL,
   …
   PARTITION BY (AREACODE_NO ASC)
   …
```

```
CREATE INDEX AREACODE_IX1 ON AREA_CODES (AREACODE_NO)
  CLUSTER  (…
    PARTITION 2 ENDING AT (400),
    PARTITION 3 ENDING AT (500),
    PARTITION 4 ENDING AT (600)),
    …);
```

GUPI

The following figure illustrates the partitioning index on the AREA_CODES table.

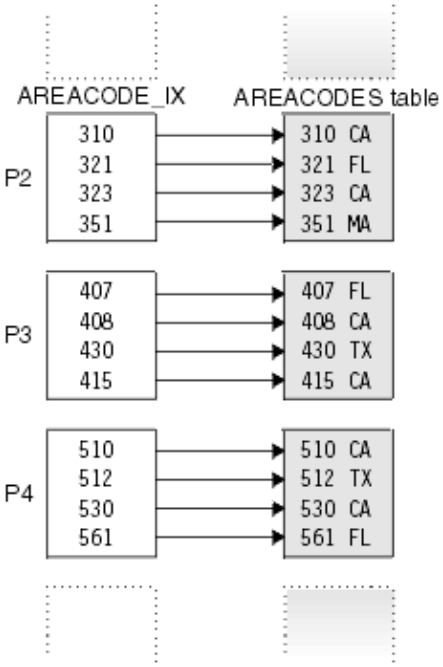


Figure 19. Partitioning index on the AREA_CODES table

Related information:

Creating tables partitioned by data value ranges (Db2 Administration Guide)
"Creation of a table with table-controlled partitioning" on page 94
Changing the boundary between partitions (Db2 Administration Guide)
"Clustering indexes" on page 108
Converting table spaces to use table-controlled partitioning (Db2 Administration Guide)

Nonpartitioning index (secondary index)

An index that is not a partitioning index is a *nonparitioning index* or *secondary index*. You can create a secondary index on a table to enforce a unique constraint, to cluster data, or to provide access paths to data for queries.

The usefulness of an index depends on the columns in its key and the cardinality of the key. Columns that you frequently select, join, group, or order are good candidates for keys. In addition, the number of distinct values in an index key for a large table must be sufficient for Db2 to use the index to retrieve data. Otherwise, Db2 might choose to do a table space scan.

You can create two types of secondary indexes: those that are partitioned (called data-partitioned secondary indexes) and those that are nonpartitioned (called nonpartitioned secondary indexes).

Data-partitioned secondary index (DPSI)

A *data-partitioned secondary index* (DPSI) is a nonpartitioning index that is physically partitioned according to the partitioning scheme of the underlying data.

A DPSI has as many partitions as the number of partitions in the table space. Each DPSI partition contains keys for the rows of the corresponding table space partition only. For example, if the table space has three partitions, the keys in DPSI partition 1 reference only the rows in table space partition 1; the keys in DPSI partition 2 reference only the rows in table space partition 2, and so on.

Restrictions:

- You can create a DPSI only on a table in a partitioned table space.
- You cannot create a DPSI for a partition-by-growth table space.
- An XML index cannot be a DPSI.

To define a DPSI, use the PARTITIONED keyword in the CREATE INDEX statement and specify an index key that does not match the partitioning key columns. If the leftmost columns of the index that you specify with the PARTITIONED keyword match the partitioning key, Db2 creates the index as a DPSI only if the collating sequence of the matching columns is different.

The use of DPSIs promotes partition independence and therefore provides the following performance advantages, among others:

- Eliminates contention between parallel LOAD utility jobs with the PART option that target different partitions of a table space
- Facilitates partition-level operations such as adding a partition or rotating a partition to be the last partition
- Improves the recovery time of secondary indexes on partitioned table spaces

However, the use of DPSIs does not always improve the performance of queries. For example, for queries with predicates that reference only the columns in the key of the DPSI, Db2 must probe each partition of the index for values that satisfy the predicate.

DPSIs provide performance advantages for queries that meet all of the following criteria:

- The query has predicates on the DPSI columns.
- The query contains additional predicates on the partitioning columns of the table that limit the query to a subset of the partitions in the table.

Nonpartitioned secondary index (NPI or NPSI)

A *nonpartitioned secondary index* (NPI or NPSI) is any index that is not defined as a partitioning index or a partitioned index. An NPI index has one index space that contains keys for the rows of all partitions of the table space.

You can create an NPI on a table in a partitioned table space. These indexes do not apply to nonpartitioned table spaces.

NPIs provide performance advantages for queries that meet the following criteria:

- The query does not contain predicates on the partitioning columns of the table that limit the query to a small subset of the partitions in the table.
- The query qualifications match the index columns.
- The SELECT list columns are included in the index (for index-only access).

GUPI

To understand the advantages of using DPSIs and NPIs, consider the following example. Suppose that you use the following SQL statements to create a DPSI (DPSIIX2) and an NPI (NPSIIX3) on the AREA_CODES table.

```
CREATE INDEX DPSIIX2 ON AREA_CODES (STATE) PARTITIONED;
CREATE INDEX NPSIIX3 ON AREA_CODES (STATE);
```

The AREA_CODES table must be partitioned on something other than the STATE column for these indexes to be secondary indexes.

The following figure illustrates what these indexes look like.

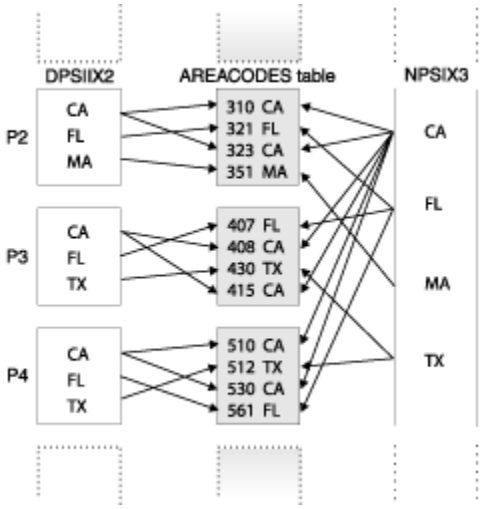


Figure 20. DPSI and NPI on AREA_CODES table

Consider the following SELECT statement:

```
SELECT STATE FROM AREA_CODES
 WHERE AREACODE_NO *<= 300 AND STATE = ‘CA';
```

This query can make efficient use of the DPSI. The number of key values that need to be searched is limited to the key values of the qualifying partitions. If a nonpartitioned secondary query, there may be a more comprehensive index scan of the key values.

Consider the following SELECT statement:

```
SELECT STATE FROM AREA_CODES
 WHERE AREACODE_NO <= 300 AND STATE > ‘CA';
```

This query makes efficient use of the NPI on columns AREACODE_NO and STATE, partitioned by STATE. The number of key values that need to be searched is limited to scanning the index key values that are lower than or equal to 300.

GUPI

DPSIs provide advantages over NPIs for utility processing. For example, utilities such as COPY, REBUILD INDEX, and RECOVER INDEX can operate on physical partitions rather than logical partitions because the keys for a data partition reside in a single DPSI partition. This method can provide greater availability.

Related concepts

Page range screening (PAGE_RANGE='Y') (Db2 Performance)

Efficient queries for tables with data-partitioned secondary indexes (Db2 Performance)

Table space scan access (ACCESSTYPE='R' and PREFETCH='S') (Db2 Performance)

Related tasks

Designing indexes for performance (Db2 Performance)

Creation of views

When you design your database, you might need to give users access to only certain pieces of data. You can give users controlled access by designing and using views.

Use the CREATE VIEW statement to define and name a view. Unless you specifically list different column names after the view name, the column names of the view are the same as the column names of the underlying table. When you create different column names for your view, remember the naming conventions that you established when designing the relational database.

A SELECT statement describes the information in the view. The SELECT statement can name other views and tables, and it can use the WHERE, GROUP BY, and HAVING clauses. It cannot use the ORDER BY clause or name a host variable.

Related concepts

Db2 views

A *view* is an alternative way of representing data that exists in one or more tables. A view can include all or some of the columns from one or more base tables.

Customized data views

A view offers an alternative way of describing data that exists in one or more tables.

A view on a single table

You can create views on individual tables when you need to limit access to particular columns.

GUPI

Example: Assume that you want to create a view on the DEPT table. Of the four columns in the table, the view needs only three: DEPTNO, DEPTNAME, and MGRNO. The order of the columns that you specify in the SELECT clause is the order in which they appear in the view:

```
CREATE VIEW MYVIEW AS
  SELECT DEPTNO,DEPTNAME,MGRNO
  FROM DEPT;
```

Example: In the preceding example, no column list follows the view name, MYVIEW. Therefore, the columns of the view have the same names as those of the DEPT table on which it is based. You can execute the following SELECT statement to see the view contents:

```
SELECT * FROM MYVIEW;
```

The result table looks like this:

```
DEPTNO     DEPTNAME                 MGRNO
======     ====================     ======
A00        CHAIRMANS OFFICE         000010
B01        PLANNING                 000020
C01        INFORMATION CENTER       000030
D11        MANUFACTURING SYSTEMS    000060
E21        SOFTWARE SUPPORT         ------
```

GUPI

A view that combines information from several tables

You can create a view that contains a union of more than one table. A union of more than one table is called a *join*.

Db2 provides two types of joins—an outer join and an inner join. An *outer join* includes rows in which the values in the join columns don't match, and rows in which the values match. An *inner join* includes only rows in which matching values in the join columns are returned.

GUPI

Example: The following example is an inner join of columns from the DEPT and EMP tables. The WHERE clause limits the view to just those columns in which the MGRNO in the DEPT table matches the EMPNO in the EMP table:

```
CREATE VIEW MYVIEW  AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
  FROM DEPT, EMP
  WHERE EMP.EMPNO = DEPT.MGRNO;
```

The result of executing this CREATE VIEW statement is an inner join view of two tables, which is shown below:

```
DEPTNO     MGRNO     LASTNAME     ADMRDEPT
======     ======    ========     ========
A00        000010    HAAS              A00
B01        000020    THOMPSON          A00
C01        000030    KWAN              A00
D11        000060    STERN             D11
```

Example: Suppose that you want to create the view in the preceding example, but you want to include only those departments that report to department A00. Suppose also that you prefer to use a different set of column names. Use the following CREATE VIEW statement:

```
CREATE VIEW MYVIEWA00
  (DEPARTMENT, MANAGER, EMPLOYEE_NAME, REPORT_TO_NAME)
  AS
  SELECT DEPTNO, MGRNO, LASTNAME, ADMRDEPT
  FROM EMP, DEPT
  WHERE EMP.EMPNO = DEPT.MGRNO
  AND ADMRDEPT = 'A00';
```

You can execute the following SELECT statement to see the view contents:

```
SELECT * FROM MYVIEWA00;
```

When you execute this SELECT statement, the result is a view of a subset of the same data, but with different column names, as follows:

```
DEPARTMENT     MANAGER        EMPLOYEE_NAME         REPORT_TO_NAME
==========     =======        =============         ==============
A00            000010         HAAS                  A00
B01            000020         THOMPSON              A00
C01            000030         KWAN                  A00
```

GUPI

Related concepts

Ways to merge lists of values

There are several ways to use the UNION keyword for merging lists of values.

Inserts and updates of data through views

If you define a view on a single table, you can refer to the name of a view in insert, update, or delete operations. If the view is complex or involves multiple tables, you must define an INSTEAD OF trigger before that view can be referenced in an INSERT, UPDATE, MERGE, or DELETE statement. This information explains how the simple case is dealt with, where Db2 makes an insert or update to the base table.

To ensure that the insert or update conforms to the view definition, specify the WITH CHECK OPTION clause. The following example illustrates some undesirable results of omitting that check.

GUPI

Example: Suppose that you define a view, V1, as follows:

```
CREATE VIEW V1 AS
  SELECT * FROM EMP
  WHERE DEPT LIKE 'D%'
```

A user with the SELECT privilege on view V1 can see the information from the EMP table for employees in departments whose IDs begin with D. The EMP table has only one department (D11) with an ID that satisfies the condition.

Assume that a user has the INSERT privilege on view V1. A user with both SELECT and INSERT privileges can insert a row for department E01, perhaps erroneously, but cannot select the row that was just inserted.

The following example shows an alternative way to define view V1.

Example: You can avoid the situation in which a value that does not match the view definition is inserted into the base table. To do this, instead define view V1 to include the WITH CHECK OPTION clause:

```
CREATE VIEW V1 AS SELECT * FROM EMP
  WHERE DEPT LIKE 'D%' WITH CHECK OPTION;
```

With the new definition, any insert or update to view V1 must satisfy the predicate that is contained in the WHERE clause: DEPT LIKE 'D%'. The check can be valuable, but it also carries a processing cost; each potential insert or update must be checked against the view definition. Therefore, you must weigh the advantage of protecting data integrity against the disadvantage of the performance degradation.

GUPI

Related tasks

Inserting, updating, and deleting data in views by using INSTEAD OF triggers (Db2 Application programming and SQL)

Changing data by using views that reference temporal tables (Db2 Administration Guide)

Related reference

CREATE VIEW (Db2 SQL)

Creation of large objects

Defining large objects to Db2 is different than defining other types of data and objects.

These are the basic steps for defining LOBs and moving the data into Db2:

1. Define a column of the appropriate LOB type.

   When you create a table with a LOB column, or alter a table to add a LOB column, defining a ROWID column is optional. If you do not define a ROWID column, Db2 defines a hidden ROWID column for you. Define only one ROWID column, even if multiple LOB columns are in the table.

   The LOB column holds information about the LOB, not the LOB data itself. The table that contains the LOB information is called the *base table*, which is different from the common base table. Db2 uses the ROWID column to locate your LOB data. You can define the LOB column and the ROWID column in a CREATE TABLE or ALTER TABLE statement. If you are adding a LOB column and a ROWID column to an existing table, you must use two ALTER TABLE statements. If you add the ROWID after you add the LOB column, the table has two ROWIDs; a hidden one and the one that you created. Db2 ensures that the values of the two ROWIDs are always the same.

2. Create a table space and table to hold the LOB data.

   For LOB data, the table space is called a LOB table space, and a table is called an auxiliary table. If your base table is nonpartitioned, you must create one LOB table space and one auxiliary table for each LOB column. If your base table is partitioned, you must create one LOB table space and one

auxiliary table for each LOB column in each partition. For example, you must create three LOB table spaces and three auxiliary tables for each LOB column if your base table has three partitions. Create these objects by using the CREATE LOB TABLESPACE and CREATE AUXILIARY TABLE statements.

3. Create an index on the auxiliary table.

   Each auxiliary table must have exactly one index in which each index entry refers to a LOB. Use the CREATE INDEX statement for this task.

4. Put the LOB data into Db2.

   If the total length of a LOB column and the base table row is less than 32 KB, you can use the LOAD utility to put the data in Db2. You can also use SQL to put LOB data into Db2 that is less than 32KB. Even though the data resides in the auxiliary table, the LOAD utility statement or SQL statement that changes data specifies the base table. Using INSERT or MERGE statements can be difficult because your application needs enough storage to hold the entire value that goes into the LOB column.

GUPI

Example: Assume that you must define a LOB table space and an auxiliary table to hold employee resumes. You must also define an index on the auxiliary table. You must define the LOB table space in the same database as the associated base table. Assume that EMP_PHOTO_RESUME is a base table. This base table has a LOB column named EMP_RESUME. You can use statements like this to define the LOB table space, the auxiliary table space, and the index:

```
CREATE LOB TABLESPACE RESUMETS
  IN MYDB
  LOG NO;
COMMIT;
CREATE AUXILIARY TABLE EMP_RESUME_TAB
  IN MYDB.RESUMETS
  STORES EMP_PHOTO_RESUME
  COLUMN EMP_RESUME;
CREATE UNIQUE INDEX XEMP_RESUME
  ON EMP_RESUME_TAB;
COMMIT;
```

GUPI

You can use the LOG clause to specify whether changes to a LOB column in the table space are to be logged. The LOG NO clause in the preceding CREATE LOB TABLESPACE statement indicates that changes to the RESUMETS table space are not to be logged.

Related concepts

Large objects (LOBs) (Db2 SQL)

Related tasks

Storing LOB data in a table (Db2 Application programming and SQL)

Related reference

CREATE LOB TABLESPACE (Db2 SQL)

CREATE AUXILIARY TABLE (Db2 SQL)

ALTER TABLE (Db2 SQL)

CREATE TABLE (Db2 SQL)

Creation of databases

When you define a Db2 database, you name an eventual collection of tables, associated indexes, and the table spaces in which they are to reside.

When you decide whether to define a new database for a new set of objects or use an existing database, consider the following facts:

- You can start and stop an entire database as a unit. You can display the status of all objects in the database by using a single command that names only the database. Therefore, place a set of related tables into the same database. (The same database holds all indexes on those tables.)

- If you want to improve concurrency and memory use, keep the number of tables in a single database relatively small (maximum of 20 tables). For example, with fewer tables, Db2 performs a reorganization in a shorter length of time.
- Having separate databases allows data definitions to run concurrently and also uses less space for control blocks.

To create a database, use the CREATE DATABASE statement. A name for a database is an unqualified identifier of up to eight characters. A Db2 database name must not be the same as the name of any other Db2 database.

Implicit creation of databases

If you do not specify the IN clause in a CREATE TABLE statement, Db2 implicitly creates a database.

The name of the database is DSN*xxxxx*, where *xxxxx* is the next five-digit number from a sequence, for example:

DSN00001, DSN00002, DSN00003, ...

However, if you do not specify the IN clause in a CREATE TABLESACE statement, the table space is created in database DSNDB04.

Example

GUPI The following example shows a valid database name:

Object
Name

Database
MYDB

This CREATE DATABASE statement creates the database MYDB:

```
CREATE DATABASE MYDB
  STOGROUP MYSTOGRP
  BUFFERPOOL BP8K4
  INDEXBP BP4;
```

The STOGROUP, BUFFERPOOL, and INDEXBP clauses that this example shows establish default values. You can override these values on the definitions of the table space or index space. GUPI

Related reference

CREATE DATABASE (Db2 SQL)

Creation of relationships with referential constraints

Referential integrity is a condition in which all intended references from data in one table column to data in another table column are valid. By using referential constraints, you can define relationships between entities that you define in Db2.

Organizations that choose to enforce referential constraints have at least one thing in common. They need to ensure that values in one column of a table are valid with respect to other data values in the database.

Examples:

- A manufacturing company wants to ensure that each part in a PARTS table identifies a product number that equals a valid product number in the PRODUCTS table.
- A company wants to ensure that each value of DEPT in the EMP table equals a valid DEPTNO value in the DEPT table.

If the DBMS did not support referential integrity, programmers would need to write and maintain application code that validates the relationship between the columns. Some programs might not enforce business rules, even though it is recommended.

This programming task can be complex because of the need to make sure that only valid values are inserted or updated in the columns. When the DBMS supports referential integrity, as Db2 does, programmers avoid some complex programming tasks and can be more productive in their other work.

Related concepts

Application of business rules to relationships
Whether a given relationship is one-to-one, one-to-many, many-to-one, or many-to-many, your relationships need to make good business sense.

Related tasks

Creating tables for data integrity (Db2 Application programming and SQL)
Altering a table for referential integrity (Db2 Administration Guide)
Using referential integrity for data consistency (Managing Security)

How Db2 enforces referential constraints

This information describes what Db2 does to maintain referential integrity.

You define referential constraints between a foreign key and its parent key. Before you start to define the referential relationships and constraints, you should understand what Db2 does to maintain referential integrity. You should understand the rules that Db2 follows when users attempt to modify information in columns that are involved in referential constraints.

To maintain referential integrity, Db2 enforces referential constraints in response to any of the following events:

- An insert to a dependent table
- An update to a parent table or dependent table
- A delete from a parent table
- Running the CHECK DATA utility or the LOAD utility on a dependent table with the ENFORCE CONSTRAINTS option

When you define the constraints, you have the following choices:

CASCADE
Db2 propagates the action to the dependents of the parent table.

NO ACTION
An error occurs, and Db2 takes no action.

RESTRICT
An error occurs, and Db2 takes no action.

SET NULL
Db2 places a null value in each nullable column of the foreign key that is in each dependent of the parent table.

Db2 does not enforce referential constraints in a predefined order. However, the order in which Db2 enforces constraints can affect the result of the operation. Therefore, you should be aware of the restrictions on the definition of delete rules and on the use of certain statements. The restrictions relate to the following SQL statements: CREATE TABLE, ALTER TABLE, INSERT, UPDATE, MERGE, and DELETE.

You can use the NOT ENFORCED option of the referential constraint definition in a CREATE TABLE or ALTER TABLE statement to define an informational referential constraint. You should use this type of referential constraint only when an application process verifies the data in a referential integrity relationship.

Insert rules

The insert rules for referential integrity apply to parent and dependent tables.

The following insert rules for referential integrity apply to parent and dependent tables:

- **For parent tables:** You can insert a row at any time into a parent table without taking any action in the dependent table. For example, you can create a new department in the DEPT table without making any

change to the EMP table. If you are inserting rows into a parent table that is involved in a referential constraint, the following restrictions apply:

  - A unique index must exist on the parent key.
  - You cannot enter duplicate values for the parent key.
  - You cannot insert a null value for any column of the parent key.
- **For dependent tables:** You cannot insert a row into a dependent table unless a row in the parent table has a parent key value that equals the foreign key value that you want to insert. You can insert a foreign key with a null value into a dependent table (if the referential constraint allows this), but no logical connection exists if you do so. If you insert rows into a dependent table, the following restrictions apply:
  - Each nonnull value that you insert into a foreign key column must be equal to some value in the parent key.
  - If any field in the foreign key is null, the entire foreign key is null.
  - If you drop the index that enforces the parent key of the parent table, you cannot insert rows into either the parent table or the dependent table.

Example: Your company doesn't want to have a row in the PARTS table unless the PROD# column value in that row matches a valid PROD# in the PRODUCTS table. The PRODUCTS table has a primary key on PROD#. The PARTS table has a foreign key on PROD#. The constraint definition specifies a RESTRICT constraint. Every inserted row of the PARTS table must have a PROD# that matches a PROD# in the PRODUCTS table.

Update rules

The update rules for referential integrity apply to parent and dependent tables.

The following update rules for referential integrity apply to parent and dependent tables:

- **For parent tables:** You cannot change a parent key column of a row that has a dependent row. If you do, the dependent row no longer satisfies the referential constraint, so Db2 prohibits the operation.
- **For dependent tables:** You cannot change the value of a foreign key column in a dependent table unless the new value exists in the parent key of the parent table.

Example: When an employee transfers from one department to another, the department number for that employee must change. The new value must be the number of an existing department, or it must be null. You should not be able to assign an employee to a department that does not exist. However, in the event of a company reorganization, employees might temporarily not report to a valid department. In this case, a null value is a possibility.

If an update to a table with a referential constraint fails, Db2 rolls back all changes that were made during the update.

Delete rules

Delete rules, which are applied to parent and dependent tables, are an important part of Db2 referential integrity.

The following delete rules for referential integrity apply to parent and dependent tables:

For parent tables
For any particular relationship, Db2 enforces delete rules that are based on the choices that you specify when you define the referential constraint.

For dependent tables
At any time, you can delete rows from a dependent table without acting on the parent table.

To delete a row from a table that has a parent key and dependent tables, you must obey the delete rules for that table. To succeed, the DELETE must satisfy all delete rules of all affected relationships. The DELETE fails if it violates any referential constraint.

Example 1: Consider the parent table in the department-employee relationship. Suppose that you delete the row for department C01 from the DEPT table. That deletion affects the information in the EMP table about Sally Kwan, Heather Nicholls, and Kim Natz, who work in department C01.

Example 2: Consider the dependent in the department-employee relationship. Assume that an employee retires and that a program deletes the row for that employee from the EMP table. The DEPT table is not affected.

Construction of a referential structure

When you build a referential structure, you need to create a set of tables and indexes in the correct order.

During logical design, you express one-to-one relationships and one-to-many relationships as if the relationships are bi-directional. For example:

- An employee has a resume, and a resume belongs to an employee (one-to-one relationship).
- A department has many employees, and each employee reports to a department (one-to-many relationship).

During physical design, you restate the relationship so that it is unidirectional; one entity becomes an implied parent of the other. In this case, the employee is the parent of the resume, and the department is the parent of the assigned employees.

During logical design, you express many-to-many relationships as if the relationships are both bidirectional and multivalued. During physical design, database designers resolve many-to-many relationships by using an associative table. The relationship between employees and projects is a good example of how referential integrity is built. This is a many-to-many relationship because employees work on more than one project, and a project can have more than one employee assigned.

Example: To resolve the many-to-many relationship between employees (in the EMP table) and projects (in the PROJ table), designers create a new associative table, EMP_PROJ, during physical design. EMP and PROJ are both parent tables to the child table, EMP_PROJ.

When you establish referential constraints, you must create parent tables with at least one unique key and corresponding indexes before you can define any corresponding foreign keys on dependent tables.

Related concepts

Database design with denormalization
The rules of normalization do not consider performance. In some cases, you need to consider denormalization to improve performance.

Entities for different types of relationships
In a relational database, separate entities must be defined for different types of relationships.

Related tasks

Using referential integrity for data consistency (Managing Security)

Tables in a referential structure

In a referential structure, you can create table spaces in any order. Using a model for the structure can be helpful.

You can create table spaces in any order. However, you need to create the table spaces before you perform the following steps. (This procedure uses the DEPT and EMP tables.)

GUPI

1. Create the DEPT table and define its primary key on the DEPTNO column. The PRIMARY KEY clause of the CREATE TABLE statement defines the primary key.

   Example:

   ```
   CREATE TABLE DEPT
     ⋮
     PRIMARY KEY (DEPTNO);
   ```

2. Create the EMP table and define its primary key as EMPNO and its foreign key as DEPT. The FOREIGN KEY clause of the CREATE TABLE statement defines the foreign key.

   Example:

```
CREATE TABLE EMP
  ⋮
  PRIMARY KEY (EMPNO)
  FOREIGN KEY (DEPT)
    REFERENCES DEPT (DEPTNO)
    ON DELETE SET NULL;
```

3. Alter the DEPT table to add the definition of its foreign key, MGRNO.

   Example:

```
ALTER TABLE DEPT
  FOREIGN KEY (MGRNO)
    REFERENCES EMP (EMPNO)
    ON DELETE RESTRICT;
```

GUPI

Related tasks

Using referential integrity for data consistency (Managing Security)

Creation of exception tables

Before you load tables that are involved in a referential constraint or check constraint, you need to create exception tables. An *exception table* contains the rows that the CHECK DATA utility identified because they violate referential constraints or check constraints.

Related reference

Exception tables for the CHECK DATA utility (Db2 Utilities)

Creation of triggers

You can use triggers to define and enforce business rules that involve different states of the data. Triggers automatically execute a set of SQL statements whenever a specified event occurs. These statements validate and edit database changes, read and modify the database, and invoke functions that perform various operations.

Triggers are optional. You define triggers by using the CREATE TRIGGER statement.

Example: Assume that the majority of your organization's salary increases are less than or equal to 10 percent. Assume also that you need to receive notification of any attempts to increase a value in the salary column by more than that amount. To enforce this requirement, Db2 compares the value of a salary before a salary increase to the value that would exist after a salary increase. You can use a trigger in this case. Whenever a program updates the salary column, Db2 activates the trigger. In the triggered action, you can specify that Db2 is to perform the following actions:

- Update the value in the salary column with a valid value, rather than preventing the update altogether.
- Notify an administrator of the attempt to make an invalid update.

As a result of using a trigger, the notified administrator can decide whether to override the original salary increase and allow a larger-than-normal salary increase.

Recommendation: For rules that involve only one condition of the data, consider using referential constraints and check constraints rather than triggers.

Triggers also move the application logic that is required to enforce business rules into the database, which can result in faster application development and easier maintenance. In the previous example, which limits salary increases, the logic is in the database, rather than in an application. Db2 checks the validity of the changes that any application makes to the salary column. In addition, if the logic ever changes (for example, to allow 12 percent increases), you don't need to change the application programs.

Related concepts

Triggers

A *trigger* defines a set of actions that are executed when a delete, insert, or update operation occurs on a specified table or view. When such an operation is executed, the trigger is said to be *activated*.

Related reference
CREATE TRIGGER (Db2 SQL)

Creation of user-defined functions

You can create your own functions in Db2 to simplify your queries.

There are three primary types of user-defined functions.

Sourced functions
Functions that are based on existing functions.

External functions
Functions that are developed by users.

SQL functions
Functions that are defined to the database by use of SQL statements only.

External user-defined functions can return a single value or a table of values.

- External functions that return a single value are called *user-defined scalar functions*.
- External functions that return a table are called *user-defined table functions*.

User-defined functions, like built-in functions or operators, support the manipulation of distinct types.

GUPI

The following two examples demonstrate how to define and use both a user-defined function and a distinct type.

Example 1: Suppose that you define a table called EUROEMP. One column of this table, EUROSAL, has a distinct type of EURO, which is based on DECIMAL(9,2). You cannot use the built-in AVG function to find the average value of EUROSAL because AVG operates on built-in data types only. You can, however, define an AVG function that is sourced on the built-in AVG function and accepts arguments of type EURO:

```
CREATE FUNCTION AVG(EURO)
  RETURNS EURO
  SOURCE SYSIBM.AVG(DECIMAL);
```

Example 2: You can then use this function to find the average value of the EUROSAL column:

```
SELECT AVG(EUROSAL) FROM EUROEMP;
```

The next two examples demonstrate how to define and use an external user-defined function.

Example 3: Suppose that you define and write a function, called REVERSE, to reverse the characters in a string. The definition looks like the following example:

```
CREATE FUNCTION REVERSE(VARCHAR(100))
  RETURNS VARCHAR(100)
  EXTERNAL NAME 'REVERSE'
  PARAMETER STYLE SQL
  LANGUAGE C;
```

Example 4: You can then use the REVERSE function in an SQL statement wherever you would use any built-in function that accepts a character argument, as shown in the following example:

```
SELECT REVERSE(:CHARSTR)
  FROM SYSDUMMY1;
```

Although you cannot write user-defined aggregate functions, you can define sourced user-defined aggregate functions that are based on built-in aggregate functions. This capability is useful in cases where you want to refer to an existing user-defined function by another name or where you want to pass a distinct type.

The next two examples demonstrate how to define and use a user-defined table function.

Example 5: You can define and write a user-defined table function that users can invoke in the FROM clause of a SELECT statement. For example, suppose that you define and write a function called BOOKS. This function returns a table of information about books on a specified subject. The definition looks like the following example:

```
CREATE FUNCTION BOOKS     (VARCHAR(40))
  RETURNS TABLE (TITLE_NAME     VARCHAR(25),
                 AUTHOR_NAME    VARCHAR(25),
                 PUBLISHER_NAME VARCHAR(25),
                 ISBNNO         VARCHAR(20),
                 PRICE_AMT      DECIMAL(5,2),
                 CHAP1_TXT      CLOB(50K))
  LANGUAGE COBOL
  PARAMETER STYLE SQL
  EXTERNAL NAME BOOKS;
```

Example 6: You can then include the BOOKS function in the FROM clause of a SELECT statement to retrieve the book information, as shown in the following example:

```
SELECT B.TITLE_NAME, B.AUTHOR_NAME, B.PUBLISHER_NAME, B.ISBNNO
  FROM TABLE(BOOKS('Computers')) AS B
  WHERE B.TITLE_NAME LIKE '%COBOL%';
```

GUPI

Related concepts

User-defined functions

User-defined functions are small programs that you can write to perform an operation. You can use a user-defined function wherever you can use a built-in function.

Sample user-defined functions (Db2 SQL)

Related tasks

Creating a user-defined function (Db2 Application programming and SQL)

Creating user-defined functions (Db2 Administration Guide)

Controlling user-defined functions (Db2 Administration Guide)

Application processes, concurrency, and recovery

All SQL programs execute as part of an *application process*. An application process involves the execution of one or more programs, and it is the unit to which Db2 allocates resources and locks.

Different application processes might involve the execution of different programs, or different executions of the same program. The means of initiating and terminating an application process are dependent on the environment.

Locking, commit, and rollback

More than one application process might request access to the same data at the same time. Furthermore, under certain circumstances, an SQL statement can execute concurrently with a utility on the same table space. *Locking* is used to maintain data integrity under such conditions, preventing, for example, two application processes from updating the same row of data simultaneously.

Db2 implicitly acquires locks to prevent uncommitted changes made by one application process from being perceived by any other. Db2 will implicitly release all locks it has acquired on behalf of an application process when that process ends, but an application process can also explicitly request that locks be released sooner. A *commit* operation releases locks acquired by the application process and commits database changes made by the same process.

Db2 provides a way to *back out* uncommitted changes made by an application process. This might be necessary in the event of a failure on the part of an application process, or in a *deadlock* situation. An application process, however, can explicitly request that its database changes be backed out. This operation is called *rollback*.

The interface used by an SQL program to explicitly specify these commit and rollback operations depends on the environment. If the environment can include recoverable resources other than Db2 databases, the SQL COMMIT and ROLLBACK statements cannot be used. Thus, these statements cannot be used in an IMS, CICS, or WebSphere environment.

Unit of work

A *unit of work* is a recoverable sequence of operations within an application process. A unit of work is sometimes called a *logical unit of work*.

At any time, an application process has a single unit of work, but the life of an application process can involve many units of work as a result of commit or full rollback operations.

A unit of work is initiated when an application process is initiated. A unit of work is also initiated when the previous unit of work is ended by something other than the end of the application process. A unit of work is ended by a commit operation, a full rollback operation, or the end of an application process. A commit or rollback operation affects only the database changes made within the unit of work it ends. While these changes remain uncommitted, other application processes are unable to perceive them unless they are running with an isolation level of uncommitted read. The changes can still be backed out. Once committed, these database changes are accessible by other application processes and can no longer be backed out by a rollback. Locks acquired by Db2 on behalf of an application process that protects uncommitted data are held at least until the end of a unit of work.

The initiation and termination of a unit of work define *points of consistency* within an application process. A point of consistency is a claim by the application that the data is consistent. For example, a banking transaction might involve the transfer of funds from one account to another. Such a transaction would require that these funds be subtracted from the first account, and added to the second. Following the subtraction step, the data is inconsistent. Only after the funds have been added to the second account is consistency reestablished. When both steps are complete, the commit operation can be used to end the unit of work, thereby making the changes available to other application processes. The following figure illustrates this concept.

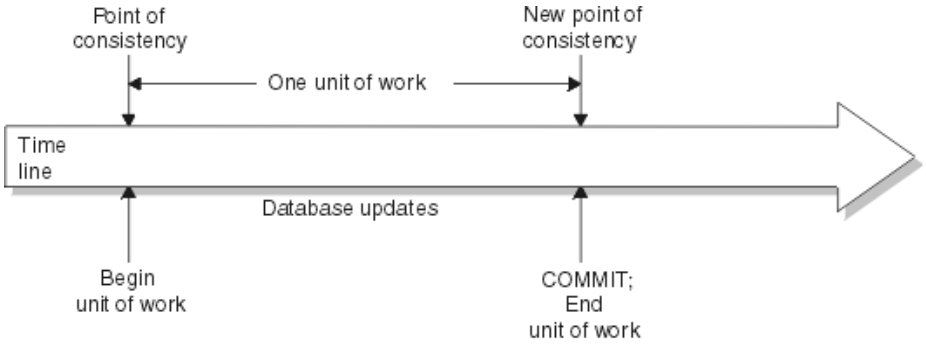


Figure 21. Unit of work with a commit operation

Unit of recovery

A *Db2 unit of recovery* is a recoverable sequence of operations executed by Db2 for an application process.

If a unit of work involves changes to other recoverable resources, the unit of work will be supported by other units of recovery. If relational databases are the only recoverable resources used by the application process, then the scope of the unit of work and the unit of recovery are the same and either term can be used.

Rolling back work

Db2 can back out all changes made in a unit of recovery or only selected changes. Only backing out all changes results in a point of consistency.

Rolling back all changes

The SQL ROLLBACK statement without the TO SAVEPOINT clause specified causes a full rollback operation. If such a rollback operation is successfully executed, Db2 backs out uncommitted changes to restore the data consistency that existed when the unit of work was initiated.

That is, Db2 undoes the work, as shown in the following figure:

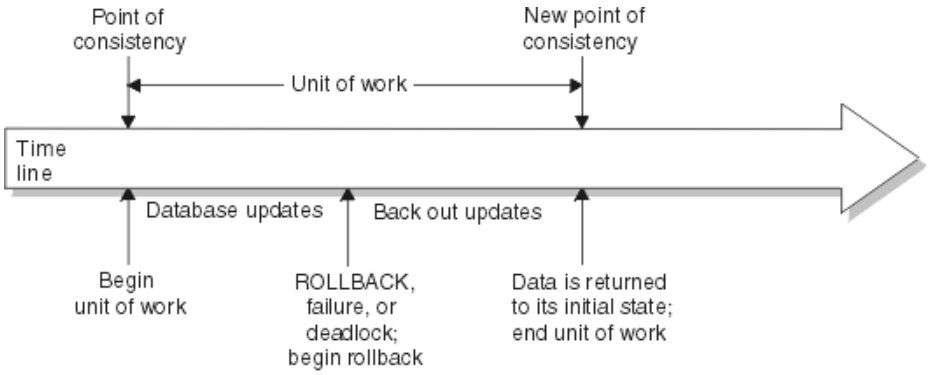


Figure 22. Rolling back all changes from a unit of work

Rolling back selected changes using savepoints

A *savepoint* represents the state of data at some particular time during a unit of work. An application process can set savepoints within a unit of work, and then as logic dictates, roll back only the changes that were made after a savepoint was set.

For example, part of a reservation transaction might involve booking an airline flight and then a hotel room. If a flight gets reserved but a hotel room cannot be reserved, the application process might want to undo the flight reservation without undoing any database changes made in the transaction prior to making the flight reservation. SQL programs can use the SQL SAVEPOINT statement to set savepoints, the SQL ROLLBACK statement with the TO SAVEPOINT clause to undo changes to a specific savepoint or the last savepoint that was set, and the SQL RELEASE SAVEPOINT statement to delete a savepoint. The following figure illustrates this concept.

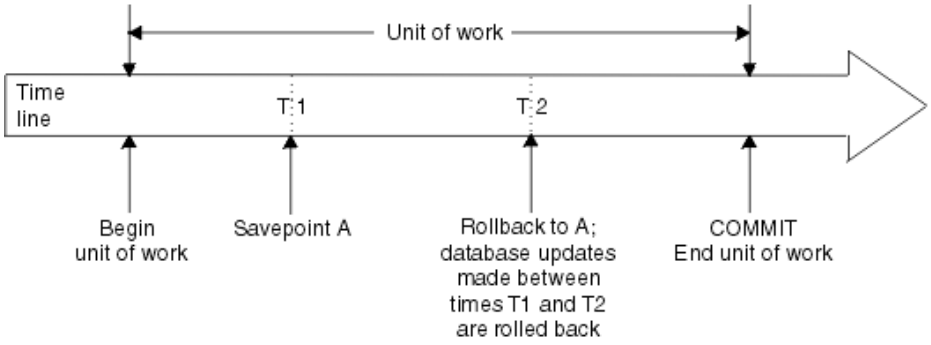


Figure 23. Rolling back changes to a savepoint within a unit of work

Packages and application plans

A *package* contains control structures that Db2 uses when it runs SQL statements. An *application plan* relates an application process to a local instance of Db2 and specifies processing options.

Packages are produced during program preparation. You can think of the control structures as the bound or operational form of SQL statements. All control structures in a package are derived from the SQL statements that are embedded in a single source program.

An application plan contains a list of package names.

Db2 applications require an application plan. Packages make application programs more flexible and easier to maintain.

In general, you create plans and packages by using the Db2 commands BIND PLAN and BIND PACKAGE.

Example

The following figure shows an application plan that contains two packages. Suppose that you decide to change the SELECT statement in package AA to select data from a different table. In this case, you need to bind only package AA again and not package AB.

GUPI

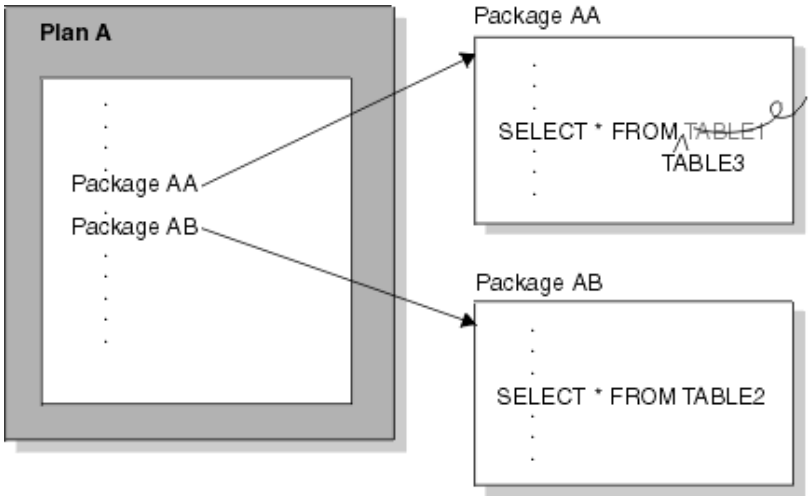


Figure 24. Application plan and packages

GUPI

Packages for triggers and routines

A *trigger package* is a special type of package that is created when you issue a CREATE TRIGGER statement. It contains the trigger body and the WHEN clause for the trigger. When the trigger activates, the package executes one or more times.

Packages are also generated, or explicitly created for certain SQL routines, including stored procedures and certain types of user defined functions. The package for a routine contains the routine body, including control statements. The package might also contain statements that are generated by Db2. Each time that the routine is invoked, the package executes one or more times.

Specifically, Db2 generates a package automatically for the following types of SQL routines:

- External scalar functions
- External table functions
- Compiled SQL scalar functions
- Native SQL procedures

- External SQL procedures

For external stored procedures, you explicitly bind the source code for the stored procedure into a package before or you issue the CREATE PROCEDURE statement. Each time that the external procedure is invoked, the package executes one or more times.

Packages for JDBC, SQLJ, and ODBC applications

Packages for JDBC, SQLJ, and ODBC applications serve different purposes. For more information, see and Db2 ODBC run time environment setup (Db2 Programming for ODBC).

Related concepts

Programming for Db2 for z/OS
You use application programs that contain SQL statements to perform data definition or data manipulation operations on Db2 for z/OS objects.

Preparation process for an application program
How you prepare an application program to run depends on the type of application. The program preparation steps for applications vary based on the type of programming language that is used.

Related reference

SET CURRENT PACKAGE PATH (Db2 SQL)
SET CURRENT PACKAGESET (Db2 SQL)
CREATE FUNCTION (Db2 SQL)
CREATE PROCEDURE (Db2 SQL)
CREATE TRIGGER (Db2 SQL)

Character conversion

A *string* is a sequence of bytes that can represent characters. Within a string, all the characters are represented by a common encoding representation. In some cases, it might be necessary to convert these characters to a different encoding representation. The process of conversion is known as *character conversion*.

Character conversion, when required, is automatic, and when successful, it is transparent to the application.

In client/server environments, character conversion can occur when an SQL statement is executed remotely. Consider, for example, the following two cases. In either case, the data could have a different representation at the sending and receiving systems.

- The values of data sent from the requester to the current server
- The values of data sent from the current server to the requester

Conversion can also occur during string operations on the same system, as in the following examples:

- An overriding CCSID is specified.

  For example, an SQL statement with a descriptor, which requires an SQLDA. In the SQLDA, the CCSID is in the SQLNAME field for languages other than REXX, and in the SQLCCSID field for REXX. (For more information, see SQL descriptor area (SQLDA) (Db2 SQL)). A DECLARE VARIABLE statement can also be issued to associate a CCSID with the host variables into which data is retrieved from a table.
- The value of the ENCODING bind option or the APPLICATION ENCODING SCHEMA option of the CREATE PROCEDURE or ALTER PROCEDURE statement for a native SQL procedure (static SQL statements) or the CURRENT APPLICATION ENCODING SCHEME special register (for dynamic SQL) is different than encoding scheme of the data being retrieved.
- A mixed character string is assigned to an SBCS column or host variable.
- An SQL statement refers to data that is defined with different CCSIDs.

The text of an SQL statement is also subject to character conversion because it is a character string.

The following list defines some of the terms used for character conversion.

ASCII
Acronym for American Standard Code for Information Interchange, an encoding scheme used to represent characters. The term ASCII is used throughout this information to refer to IBM-PC Data or ISO 8-bit data.

character set
A defined set of characters, a character being the smallest component of written language that has semantic value. For example, the following character set appears in several code pages:

- 26 nonaccented letters A through Z
- 26 nonaccented letters a through z
- digits 0 through 9
- . , : ; ? ( ) ' "/- _ & + % * = < >

code page
A set of assignments of characters to code points. For example, in EBCDIC, "A" is assigned code point X'C1', and "B" is assigned code point X'C2'. In Unicode UTF-8, "A" is assigned code point X'41', and "B" is assigned code point X'42'. Within a code page, each code point has only one specific meaning.

code point
A unique bit pattern that represents a character. It is a numerical index or position in an encoding table used for encoding characters.

coded character set
A set of unambiguous rules that establishes a character set and the one-to-one relationships between the characters of the set and their coded representations. It is the assignment of each character in a character set to a unique numeric code value.

coded character set identifier (CCSID)
A two-byte, unsigned binary integer that uniquely identifies an encoding scheme and one or more pairs of character sets and code pages.

EBCDIC
Acronym for Extended Binary-Coded Decimal Interchange Code, an encoding scheme used to represent character data, a group of coded character sets that consist of 8 bit coded characters. EBCDIC coded character sets use the first 64 code positions (X'00' to X'3F') for control codes. The range X'41' to X'FE' is used for single-byte characters. For double-byte characters, the first byte is in the range X'41' to X'FE' and the second byte is also in the range X'41' to X'FE', while X'4040' represents a double-byte space.

encoding scheme
A set of rules used to represent character data. All string data stored in a table must use the same encoding scheme and all tables within a table space must use the same encoding scheme, except for global temporary tables, declared temporary tables, and work file table spaces. Db2 supports these encoding schemes:

- ASCII
- EBCDIC
- Unicode

substitution character
A unique character that is substituted during character conversion for any characters in the source encoding representation that do not have a match in the target encoding representation.

Unicode
A universal encoding scheme for written characters and text that enables the exchange of data internationally. It provides a character set standard that can be used all over the world. It provides the ability to encode all characters used for the written languages of the world and treats alphabetic characters, ideographic characters, and symbols equivalently because it specifies a numeric value and a name for each of its characters. It includes punctuation marks, mathematical symbols, technical symbols, geometric shapes, and dingbats. Db2 supports these two encoding forms:

- UTF-8: Unicode Transformation Format, a 8 bit encoding form designed for ease of use with existing ASCII-based systems. UTF-8 can encode any of the Unicode characters. A UTF-8 character is 1,2,3, or 4 bytes in length. A UTF-8 data string can contain any combination of SBCS and MBCS data, including supplementary characters. The CCSID value for data in UTF-8 format is 1208. UTF-8 has multiple code points for spaces, including the '20'X single-byte space that Db2 uses for padding UTF-8 data.
- UTF-16: Unicode Transformation Format, a 16 bit encoding form designed to provide code values for over a million characters and a superset of UCS-2. UTF-16 can encode any of the Unicode characters. In UTF-16 encoding, characters are 2 bytes in length, except for supplementary characters, which take two 2 byte string units per character. The CCSID value for data in UTF-16 format is 1200. UTF-16 has multiple code points for spaces, including the '0020'X single-byte space that Db2 uses for padding UTF-16 data.

Character data (CHAR, VARCHAR, and CLOB) is encoded in Unicode UTF-8. Character strings are also used for mixed data (that is a mixture of single-byte characters and multi-byte characters) and for data that is not associated with any character set (called bit data). Graphic data (GRAPHIC, VARGRAPHIC, and DBCLOB) is encoded in Unicode UTF-16. For a comparison of some UTF-8 and UTF-16 code points for some sample characters, see “Character sets and code pages” on page 133. This table shows how a UTF-8 character can be 1 to 4 bytes in length, a non-supplementary UTF-16 character is 2 bytes in length, and how a supplementary character in either UTF-8 or UTF-16 takes two 2 byte code points.

Character conversion can affect the results of several SQL operations. In this information, the effects are described in:

Conversion rules for string assignment (Db2 SQL)
Conversion rules for comparisons (Db2 SQL)
Character conversion in set operations and concatenations (Db2 SQL)

Character sets and code pages

Even with the same encoding scheme, different CCSIDs exist, and the same code point can represent a different character in different CCSIDs. Furthermore, a byte in a character string does not necessarily represent a character from a single-byte character set (SBCS).

The following figure shows how a typical character set might map to different code points in two different code pages.

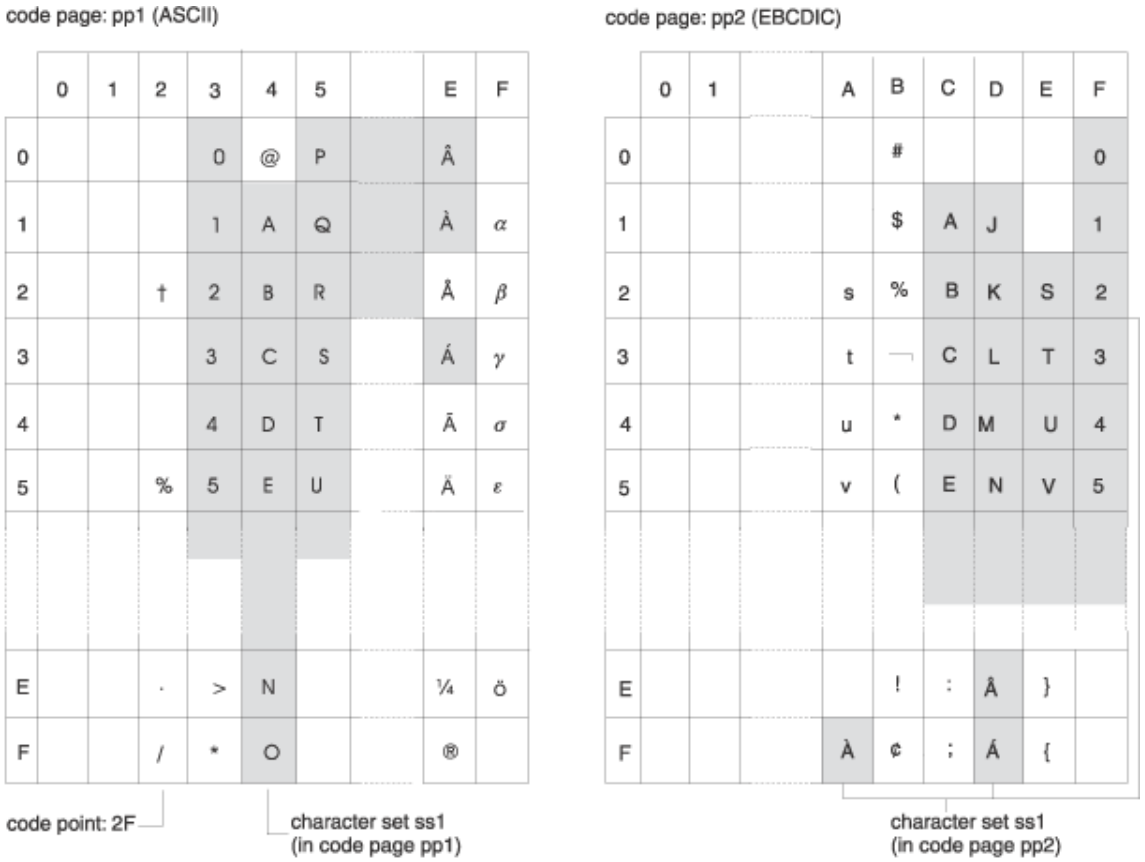


Figure 25. Code page mappings for character set ss1 in ASCII and EBCDIC

For Unicode, there is only one CCSID for UTF-8 and only one CCSID for UTF-16. The following figure shows how the first 127 single code points for UTF-8 are the same as ASCII with a CCSID of 367. For example, in both UTF-8 and ASCII CCSID 367, an A is X'41' and a 1 is X'31'.

First 127 code points for UTF-8 code page



Figure 26. Code point mapping for the first 127 code points for UTF-8 single-byte characters (CCSID 1208)

The following figure shows a comparison of how some UTF-16 and UTF-8 code points map to some sample characters. The character for the eighth note musical symbol takes two 2 byte code points because it is a supplementary character.

Character glyph	UTF-8 code point	UTF-16 code point
M	4D	004D
Ä	C384	00C4
事	E4BA8B	4E8B
♪	F09D85A0	D834DD60

Figure 27. A comparison of how some UTF-8 and UTF-16 code points map to some sample characters

Related concepts

Introduction to character conversion (Db2 Internationalization Guide)

Related information

Unicode Consortium

Coded character sets and CCSIDS

IBM's character data representation architecture (CDRA) deals with the differences in string representation and encoding. The *Coded Character Set Identifier (CCSID)* is a key element of this architecture. A CCSID is a 2 byte (unsigned) binary number that uniquely identifies an encoding scheme and one or more pairs of character sets and code pages.

A CCSID is an attribute of strings, just as length is an attribute of strings. All values of the same string column have the same CCSID.

Character conversion is described in terms of CCSIDs of the source and target. With Db2 for z/OS, two methods are used to identify valid source and target combinations and to perform the conversion from one coded character set to another:

- Db2 catalog table SYSIBM.SYSSTRINGS

  Each row in the catalog table describes a conversion from one coded character set to another.
- z/OS support for Unicode

  For more information about the conversion services that are provided, including a complete list of the IBM-supplied conversion tables, see *z/OS Support for Unicode: Using Conversion Services*.

In some cases, no conversion is necessary even though the strings involved have different CCSIDs.

Different types of conversions might be supported by each database manager. Round-trip conversions attempt to preserve characters in one CCSID that are not defined in the target CCSID so that if the data is subsequently converted back to the original CCSID, the same original characters result. Enforced subset match conversions do not attempt to preserve such characters. Which type of conversion is used for a specific source and target CCSID is product-specific.

For more information on character conversion, see *Db2 Installation Guide*.

Determining the encoding scheme and CCSID of a string

An encoding scheme and a CCSID are attributes of strings, just as length is an attribute of strings. All values of the same string column have the same encoding scheme and CCSID.

Every string has an encoding scheme and a CCSID that identifies the manner in which the characters in the string are encoded. Strings can be encoded in ASCII, EBCDIC, or Unicode.

The CCSID that is associated with a string value depends on the SQL statement in which the data is referenced and the type of expression. Use the Type 1 rules when the SQL statement meets the following conditions:

- The SQL statement operates with a single set of CCSIDs (SBCS, mixed, and graphic). An SQL statement that does not contain any of the following items operates with a single set of CCSIDs:
  - References to columns from multiple tables or views that are defined with CCSIDs from more than one set of CCSIDs (SBCS, mixed, and graphic)
  - References to an EBCDIC table that contains a Unicode column
  - Graphic hexadecimal (GX) or hexadecimal Unicode (UX) string constants
  - References to the XMLCLOB built-in function
  - Cast specifications with a CCSID clause
  - User-defined table functions
- The SQL statement is not one of the following statements:
  - CALL statement
  - SET assignment statement
  - SET special register
  - VALUES statement
  - VALUES INTO statement
- One of the following built-in functions is not referenced:
  - ASCII_CHR
  - ASCII_STR (or ASCIISTR)
  - CHR
  - DECRYPT_BIT
  - DECRYPT_CHAR
  - DECRYPT_DB
  - EBCDIC_CHR
  - EBCDIC_STR
  - GETVARIABLE
  - NORMALIZE_STRING
  - UNICODE_STR (or UNISTR)
  - XMLSERIALIZE
- The SQL statement does not include a collection-derived table (UNNEST).

Use the Type 2 rules when the statement does not meet the conditions for Type 1 rules.

For those SQL statements and tools that use a SYSDUMMY*x* table, use the SYSDUMMY*x* table that has the same encoding scheme as the other objects in your SQL statement, to avoid conversions. A SYSDUMMY*x* table is available in each encoding scheme. For example, suppose that your SQL statement references a SYSDUMMY*x* table that is in a different encoding scheme from other objects in the statement. Db2 treats this statement as one that references objects with different CCSIDs (Type 2 rules), and conversion is likely to occur. To avoid this situation, reference the SYSDUMMY*x* table that has the same encoding scheme as the other objects in your SQL statement. See "SYSDUMMYx tables" on page 339 for more information.

Table 18 on page 137 describes the rules for determining the CCSID of derived string data when the source data has a string type.

Table 18. Rules for determining the CCSID that is associated with string data derived from string data

Source of the string data	Type 1 rules	Type 2 rules
String constant	If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the string constant. The CCSID is the appropriate character string CCSID of the encoding scheme.	The CCSID is the appropriate character string CCSID of the application encoding scheme."1" on page 139
Datetime constant	If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the string constant. Otherwise, the default EBCDIC encoding scheme is used for the string constant. The CCSID is the appropriate character string CCSID of the encoding scheme.	The CCSID is the appropriate character string CCSID of the application encoding scheme."1" on page 139
Hexadecimal string constant (X'...')	If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the string constant. Otherwise, the default EBCDIC encoding scheme is used for the string constant. The CCSID is the appropriate graphic string CCSID of the encoding scheme.	The CCSID is the appropriate character string CCSID of the application encoding scheme."1" on page 139
Graphic string constant (G'...')	If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the graphic string constant. Otherwise, the default EBCDIC encoding scheme is used for the graphic string constant. The CCSID is the graphic string CCSID of the encoding scheme.	The CCSID is the graphic string CCSID of the application encoding scheme."1" on page 139
Graphic hexadecimal constant (GX'...')	Not applicable.	The CCSID is the graphic string CCSID of the application encoding scheme, which must be ASCII or EBCDIC.
Hexadecimal Unicode string constant (UX'....')	Not applicable.	The CCSID is 1200 (UTF-16).

Table 18. Rules for determining the CCSID that is associated with string data derived from string data (continued)

Source of the string data	Type 1 rules	Type 2 rules
Special register	If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the special register. Otherwise, the default EBCDIC encoding scheme is used for the special register. The CCSID is the appropriate character string CCSID of the encoding scheme.	The CCSID is the appropriate CCSID of the application encoding scheme."1" on page 139
Column of a table	The CCSID is the CCSID that is associated with the column of the table.	The CCSID is the CCSID that is associated with the column of the table.
Column of a view	The CCSID is the CCSID of the column of the result table of the fullselect of the view definition.	The CCSID is the CCSID of the column of the result table of the fullselect of the view definition.
Expression	The CCSID is the CCSID of the result of the expression.	The CCSID is the CCSID of the result of the expression.
Result of a built-in function	If the description of the function, in Built-in functions (Db2 SQL), indicates what the CCSID of the result is, the CCSID is that CCSID. Otherwise, if the description of the function refers to this table for the CCSID, the CCSID is the appropriate CCSID of the CCSID set that is used by the statement for the data type of the result.	If the description of the function, in Built-in functions (Db2 SQL), indicates what the CCSID of the result is, the CCSID is that CCSID. Otherwise, if the description of the function refers to this table for the CCSID, the CCSID is the appropriate CCSID of the application encoding scheme for the data type of the result. "1" on page 139
Parameter of a user-defined routine	The CCSID is the CCSID that was determined when the function or procedure was created.	The CCSID is the CCSID that was determined when the function or procedure was created.
The expression in the RETURN statement of a CREATE statement for a user-defined SQL scalar function	If the expression in the RETURN statement is string data, the encoding scheme is the same as for the parameters of the function. The CCSID is determined from the encoding scheme and the attributes of the data.	The CCSID is determined from the CCSID of the result of the expression specified in the RETURN statement.

Table 18. Rules for determining the CCSID that is associated with string data derived from string data (continued)

Source of the string data	Type 1 rules	Type 2 rules
String host variable	If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the host variable. Graphic variables are an exception if the table or view is EBCDIC or ASCII and the value of the MIXED DATA field on the DSNTIPF panel is NO. In this case, the Unicode encoding scheme is used for the graphic host variable. Otherwise, the default EBCDIC encoding scheme is used for the host variable. The CCSID is the appropriate CCSID of the data type of the host variable.	At package prepare time, the CCSID is the appropriate CCSID of the data type of the host variable for the application encoding scheme. Graphic variables are an exception if the application or encoding scheme is EBCDIC or ASCII and the value of the MIXED DATA field on the DSNTIPF panel is NO. In this case, the Unicode encoding scheme is used for the graphic host variable. At run time, the CCSID specified in the declare variable statement, or as an override in the SQLDA. Otherwise, the CCSID is the appropriate CCSID of the application encoding scheme for the data type of the host variable.

Notes:

1. If the context is within a check constraint or trigger package, the CCSID is the appropriate CCSID for Unicode instead of the application encoding scheme.

"Determining the encoding scheme and CCSID of a string" on page 135 shows the rules for determining the CCSID of derived string data when the source data has a numeric type.

Table 19. Rules for determining the CCSID that is associated with string data derived from numeric data

Source of the numeric data	Type 1 rules	Type 2 rules
Numeric data	If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the string data that is derived from a numeric value. Otherwise, the default EBCDIC encoding scheme is used for the string data derived from a numeric value. The CCSID is the appropriate character or graphic string CCSID of the application encoding scheme.	The CCSID is the appropriate character or graphic string CCSID of the application encoding scheme. "1" on page 139

Notes:

1. If the context is within a check constraint or trigger package, the CCSID is the appropriate CCSID for Unicode instead of the application encoding scheme.

The following examples show how these rules are applied.

Example 1: Assume that the default encoding scheme for the installation is EBCDIC and that the installation does not support mixed and graphic data. The following statement conforms to the rules for Type 1 in Table 18 on page 137. Therefore, the X'40' is interpreted as EBCDIC SBCS data because the

statement references a table that is in EBCDIC. The CCSID for X'40' is the default EBCDIC SBCS CCSID for the installation.

```
SELECT * FROM EBCDIC_TABLE WHERE COL1 = X'40';
```

the result of the query includes each row that has a value in column COL1 that is equal to a single EBCDIC blank.

Example 2: The following statement references data from two different tables that use different encoding schemes. This statement does not conform to the rules for Type 1 statements in Table 18 on page 137. Therefore, the rules for Type 2 statements are used. The CCSID for X'40' is dependent on the current application encoding scheme. Assuming that the current application encoding scheme is EBCDIC, X'40' represents a single EBCDIC blank.

```
SELECT * FROM EBCDIC_TABLE, UNICODE_TABLE WHERE COL1 = X'40';
```

as with Example 1, the result of the query includes each row that has a value in column COL1 that is equal to a single EBCDIC blank. If the current application encoding scheme were ASCII or Unicode, X'40' would represent something different and the results of the query would be different.

Expanding conversions

An *expanding conversion* occurs when the length of the converted string is greater than that of the source string.

For example, an expanding conversion occurs when an ASCII mixed data string that contains DBCS characters is converted to EBCDIC mixed data. To prevent the loss of data on expanding conversions, use a varying-length string variable with a maximum length that is sufficient to contain the expansion.

Expanding conversions also can occur when string data is converted to or from Unicode. It can also occur between UTF-8 and UTF-16, depending on the data being converted. UTF-8 uses 1, 2, 3, or 4 bytes per character. UTF-16 uses 2 bytes per character, except for supplementary characters, which use two 2 byte code points for each character. If UTF-8 were being converted to UTF-16, a 1 byte character would be expanded to 2 bytes.

Contracting conversions

A *contracting conversion* occurs when the length of the converted string is smaller than that of the source string.

For example, a contracting conversion occurs when an EBCDIC mixed data string that contains DBCS characters is converted to ASCII mixed data due to the removal of shift codes.

Contracting conversions also can occur when string data is converted to or from Unicode data. It can also occur between UTF-8 and UTF-16, depending on the data being converted.

Distributed relational databases

The database managers in a distributed relational database communicate and cooperate with each other in a way that allows a Db2 application program to use SQL to access data at any of the interconnected computer systems.

A *distributed relational database* consists of a set of tables and other objects that are spread across different, but interconnected, computer systems. Each computer system has a relational database manager, such as Db2, that manages the tables in its environment. The database managers communicate and cooperate with each other in a way that allows a Db2 application program to use SQL to access data at any of the computer systems. The Db2 subsystem where the application plan is bound is known as the *local Db2 subsystem*. Any database server other than the local Db2 subsystem is considered a *remote database server*, and access to its data is a distributed operation.

Distributed relational databases are built on formal requester-server protocols and functions. An *application requester* component supports the application end of a connection. It transforms an application's database request into communication protocols that are suitable for use in the distributed

database network. These requests are received and processed by an *application server* component at the database server end of the connection. Working together, the application requester and application server handle the communication and location considerations so that the application is isolated from these considerations and can operate as if it were accessing a local database.

For more information on Distributed Relational Database Architecture™ (DRDA) communication protocols, see *Open Group Technical Standard, DRDA Version 3 Vol. 1: Distributed Relational Database Architecture*.

Related concepts

Distributed data access
Distributed computing environments typically involve requests from users at one DBMS client that are processed by a DBMS server. The server DBMS is typically remote to the client. Certain programming techniques and performance implications apply to distributed computing environments.

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Connections

A *connection* is an association between an application process and a local or remote database server. Connections are managed by applications.

An application process must be connected to the application server facility of a database manager before SQL statements that reference tables or views can be executed. An application can use the CONNECT statement to establish a connection to a database server and make that database server the current server of the application process.

Commit processing: When Db2 for z/OS acts as a requester, it negotiates with the database server during the connection process to determine how to perform commits. If the remote server does not support two-phase commit protocol, Db2 downgrades to perform one-phase commits. Otherwise, Db2 always performs two-phase commits, which allow applications to update one or more databases in a single unit of work and are more reliable than one-phase commits. Two-phase commit is a two-step process:

1. First, all database managers involved in the same unit of work are pooled to determine whether they are ready to commit.
2. Then, if all database managers respond positively, they are directed to execute commit processing. If all database managers do not respond positively, they are directed to execute backout processing.

Db2 can also provide coordination for transactions that include both two-phase commit resources and one-phase commit resources. If an application has multiple connections to several different database servers, and if any of the connections are one-phase commit connections, then only one database that is involved in the transaction can be updated. The connections to all the other databases that are involved in the transaction are read-only.

Supported SQL statements and clauses: For the most part, an application can use the statements and clauses that are supported by the database manager of the current server, even though that application might be running via the application requester of a database manager that does not support some of those statements and clauses. Restrictions to this general rule for Db2 for z/OS are documented in *IBM Db2 SQL Reference for Cross-Platform Development*.

To execute a static SQL statement that references tables or views, the bound form of the statement is taken from a package that the database manager previously created through a bind operation or when a version of a native SQL procedure was defined.

Distributed unit of work

The *distributed unit of work facility* provides for the remote preparation and execution of SQL statements.

An application process at computer system A can connect to a database server at computer system B and, within one or more units of work, execute any number of static or dynamic SQL statements that reference objects at B. All objects referenced in a single SQL statement must be managed by the same database server. Any number of database servers can participate in the same unit of work, and any

number of connections can exist between an application process and a database server. A commit or rollback operation that does not specify a savepoint ends the unit of work.

Connection management
How connections are managed depends on what states the SQL connection and the application process are in.

At any time:

- An SQL connection is in one of the following states:
  - Current and held
  - Current and release-pending
  - Dormant and held
  - Dormant and release-pending
- An application process is in the connected or unconnected state, and has a set of zero or more SQL connections. Each SQL connection is uniquely identified by the name of the database server at the other end of the connection. Only one SQL connection is active (current) at a time.

Initial state of an application process: An application process is initially in the connected state, and it has exactly one SQL connection. The server of that connection is the local Db2 subsystem.

Initial state of an SQL connection: An SQL connection is initially in the current and held state.

The following figure shows the state transitions:

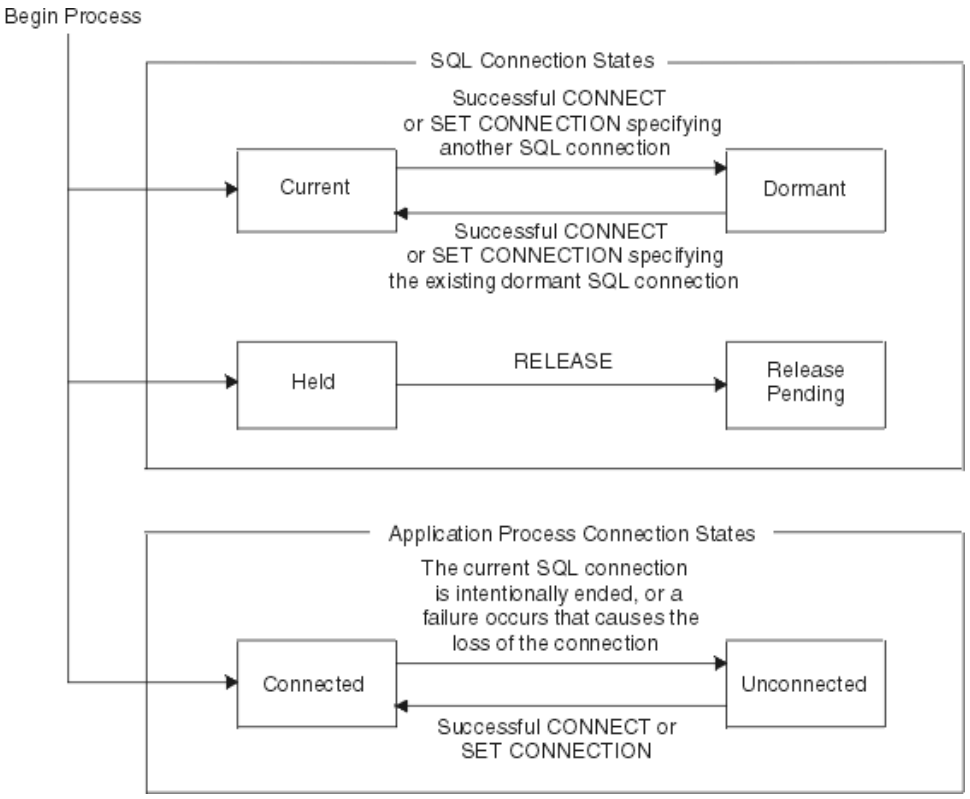


Figure 28. State transitions for an SQL connection and an application process connection in a distributed unit of work

SQL connection states
If an application process successfully executes a CONNECT statement, the SQL connection states of the connections change.

If an application process successfully executes a CONNECT statement:

- The current connection is placed in the dormant and held state.

- The new connection is placed in the current and held state.
- The location name is added to the set of existing connections.

If the location name is already in the set of existing connections, an error is returned.

An SQL connection in the dormant state is placed in the current state using:

- The SET CONNECTION statement, or
- The CONNECT statement, if the SQLRULES(Db2) bind option is in effect.

When an SQL connection is placed in the current state, the previously-current SQL connection, if any, is placed in the dormant state. No more than one SQL connection in the set of existing connections of an application process can be current at any time. Changing the state of an SQL connection from current to dormant or from dormant to current has no effect on its held or release-pending state.

An SQL connection is placed in the release-pending state by the RELEASE statement. When an application process executes a commit operation, every release-pending connection of the process is ended. Changing the state of an SQL connection from held to release-pending has no effect on its current or dormant state. Thus, an SQL connection in the release-pending state can still be used until the next commit operation. No way exists to change the state of a connection from release-pending to held.

Application process connection states
In a distributed unit of work, an application process can be in a connected or unconnected state. Depending on the state, the application process can execute only certain SQL statements successfully.

A connection to a different database server can be established by the explicit or implicit execution of a CONNECT statement. The following rules apply:

- An application process cannot have more than one SQL connection to the same database server at the same time.
- When an application process executes a SET CONNECTION statement, the specified location name must be in the set of existing connections of the application process.
- When an application process executes a CONNECT statement and the SQLRULES(STD) bind option is in effect, the specified location must not be in the set of existing connections of the application process.

If an application process has a current SQL connection, the application process is in a *connected* state. The CURRENT SERVER special register contains the name of the database server of the current SQL connection. The application process can execute SQL statements that refer to objects managed by that server. If the server is a Db2 subsystem, the application process can also execute certain SQL statements that refer to objects managed by a Db2 subsystem with which that server can establish a connection.

An application process in an unconnected state enters a connected state when it successfully executes a CONNECT or SET CONNECTION statement.

If an application process does not have a current SQL connection, the application process is in an *unconnected* state. The CURRENT SERVER special register contains blanks. The only SQL statements that can be executed successfully are CONNECT, RELEASE, COMMIT, ROLLBACK, and any of the following local SET statements.

- SET CONNECTION
- SET CURRENT APPLICATION ENCODING SCHEME
- SET CURRENT PACKAGE PATH
- SET CURRENT PACKAGESET
- SET *host-variable* = CURRENT APPLICATION ENCODING SCHEME
- SET *host-variable* = CURRENT PACKAGESET
- SET *host-variable* = CURRENT SERVER

Because the application process is in an unconnected state, a COMMIT or ROLLBACK statement is processed by the local Db2 subsystem.

An application process in a connected state enters an unconnected state when its current SQL connection is intentionally ended, or the execution of an SQL statement is unsuccessful because of a failure that causes a rollback operation at the current server and loss of the SQL connection. SQL connections are intentionally ended when an application process successfully executes a commit operation and either of the following are true:

- The SQL connection is in the release-pending state.
- The SQL connection is not in the release-pending state, but it is a remote connection and either of the following are true:
  - The DISCONNECT(AUTOMATIC) bind option is in effect
  - The DISCONNECT(CONDITIONAL) bind option is in effect and an open WITH HOLD cursor is not associated with the connection

An implicit CONNECT to a default Db2 subsystem is executed when an application process executes an SQL statement other than COMMIT, CONNECT TO, CONNECT RESET, SET CONNECTION, RELEASE, or ROLLBACK, and if all of the following conditions apply:

- The CURRENTSERVER bind option was specified when creating the application plan of the application process and the identified server is not the local Db2.
- An explicit CONNECT statement has not already been successfully or unsuccessfully executed by the application process.
- An implicit connection has not already been successfully or unsuccessfully executed by the application process. An implicit connection occurs as the result of execution of an SQL statement that contains a three-part name in a package that is bound with the DBPROTOCOL(DRDA) option.

If the implicit CONNECT fails, the application process is placed in an *unconnected* state.

When a connection is ended, all resources that were acquired by the application process through the connection and all resources that were used to create and maintain the connection are returned to the connection pool. For example, if application process *P* placed the connection to application server *X* in the release-pending state, all cursors of *P* at *X* are closed and returned to the connection pool when the connection is ended during the next commit operation.

When a connection is ended as a result of a communications failure, the application process is placed in an unconnected state.

All connections of an application process are ended when the process ends.

Remote unit of work

The *remote unit of work facility* also provides for the remote preparation and execution of SQL statements, but in a much more restricted fashion than the distributed unit of work facility.

An application process at computer system A can connect to a database server at computer system B and, within one or more units of work, execute any number of static or dynamic SQL statements that reference objects at B. All objects referenced in a single SQL statement must be managed by the same database server, and all SQL statements in the same unit of work must be executed by the same database server. However, unlike a distributed unit of work, an application process can have only one connection at a time. The process cannot connect to a new server until it executes a commit or rollback operation on the current server to end that unit of work. This restricts the situations in which a CONNECT statement can be executed.

Connection management

How connections are managed depends on what states the SQL connection and the application process are in.

An application process is in one of four states at any time:

- Connectable and connected
- Unconnectable and connected
- Connectable and unconnected

- Unconnectable and unconnected

Initial state of an application process: An application process is initially in the connectable and connected state. The database server to which the application process is connected is determined by a product-specific option that might involve an implicit CONNECT operation. An implicit connect operation cannot occur if an implicit or explicit connect operation has already successfully or unsuccessfully occurred. Thus, an application process cannot be implicitly connected to a database server more than once. Other rules for implicit connect operations are product-specific.

Figure 29 on page 145 shows the state transitions:

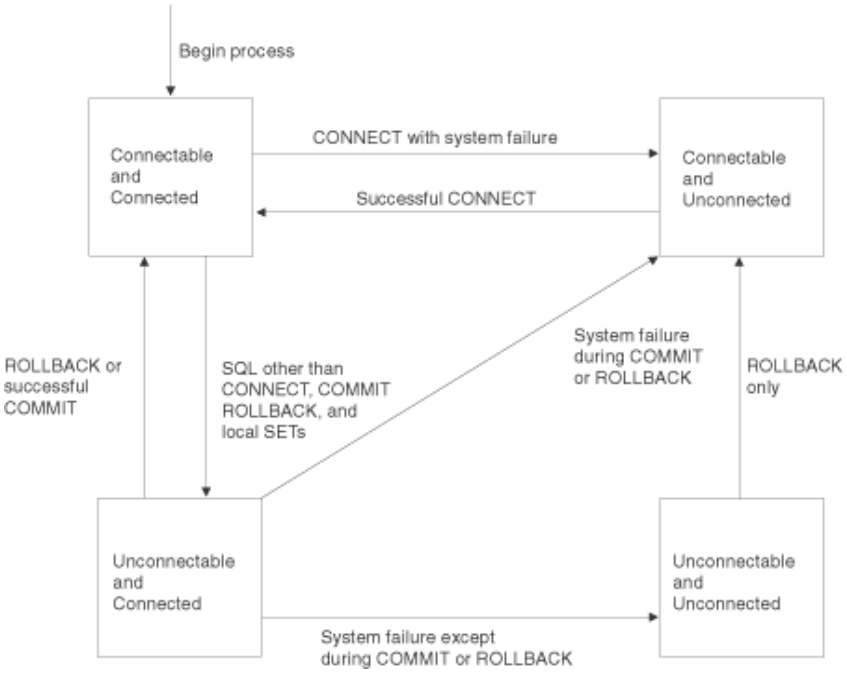


Figure 29. State transitions for an application process connection in a remote unit of work

In the following descriptions of application process connections, CONNECT can mean:

- CONNECT TO
- CONNECT RESET
- CONNECT *authorization*

It cannot mean CONNECT with no operand, which is used to return information about the current server.

Consecutive CONNECT statements can be executed successfully because CONNECT does not remove an application process from the connectable state. A CONNECT statement does not initiate a new unit of work; a unit of work is initiated by the first SQL statement that accesses data. CONNECT cannot execute successfully when it is preceded by any SQL statement other than CONNECT, COMMIT, RELEASE, ROLLBACK, or SET CONNECTION. To avoid an error, execute a commit or rollback operation before a CONNECT statement is executed.

Connectable and connected state: In the connectable and connected state, an application process is connected to a database server, and CONNECT statements that target the current server can be executed. An application process re-enters this state when either of the following is true:

- The process completes a rollback or a successful commit from an unconnectable and connected state.
- The process successfully executes a CONNECT statement from a connectable and unconnected state.

Unconnectable and connected state: In the unconnectable and connected state, an application process is connected to a database server, and only a CONNECT statement with no operands can be executed. An application process enters this state from a connectable and connected state when it executes any SQL statement other than CONNECT, COMMIT, or ROLLBACK.

Connectable and unconnected state: In the connectable and unconnected state, an application process is not connected to a database server. The only SQL statement that can be executed is CONNECT. An application process enters this state if any of the following is true:

- The process does not successfully execute a CONNECT statement from a connectable and connected state.
- The process executes a COMMIT statement when the SQL connection is in a release-pending state.
- A system failure occurs during a COMMIT or ROLLBACK from an unconnectable and connected state.
- The process executes a ROLLBACK statement from an unconnectable and unconnected state.

Other product-specific reasons can also cause an application process to enter the connectable and unconnected state.

Unconnectable and unconnected state: In the unconnectable and unconnected state, an application process is not connected to a database server and CONNECT statements cannot be executed. The only SQL statement that can be executed is ROLLBACK. An application process enters this state from an unconnectable and connected state as a result of a system failure, except during a COMMIT or ROLLBACK, at the server.

Distributed data

Distributed data is data that resides on a DBMS other than your local system.

Your *local* DBMS is the one on which you bind your package. All other DBMSs are *remote*.

Many businesses need to manage data from a wide variety of sources and locations. A distributed environment provides the flexibility that is required to allocate resources for data that is located at different sites or database management systems (DBMSs) in a computer network.

Related concepts

Distributed data access
Distributed computing environments typically involve requests from users at one DBMS client that are processed by a DBMS server. The server DBMS is typically remote to the client. Certain programming techniques and performance implications apply to distributed computing environments.

Remote servers

A remote server can be physically remote, or it can be part of the same operating system under which your local DBMS runs.

When you request services from a remote DBMS, the remote DBMS is a *server*, and your local system is a *requester* or *client*. Conceptually, a server is like a food server who takes food orders, delivers food, and provides other services to customers. The customer is like the requester, or client. The purpose of the server is to provide services to its clients.

A remote server can be truly remote in the physical sense (thousands of miles away), or a remote server can be part of the same operating system under which your local DBMS runs. This information generally assumes that your local DBMS is an instance of Db2 for z/OS. A remote server can be another instance of Db2 for z/OS or an instance of one of many other products.

The following figure shows the client/server environment.

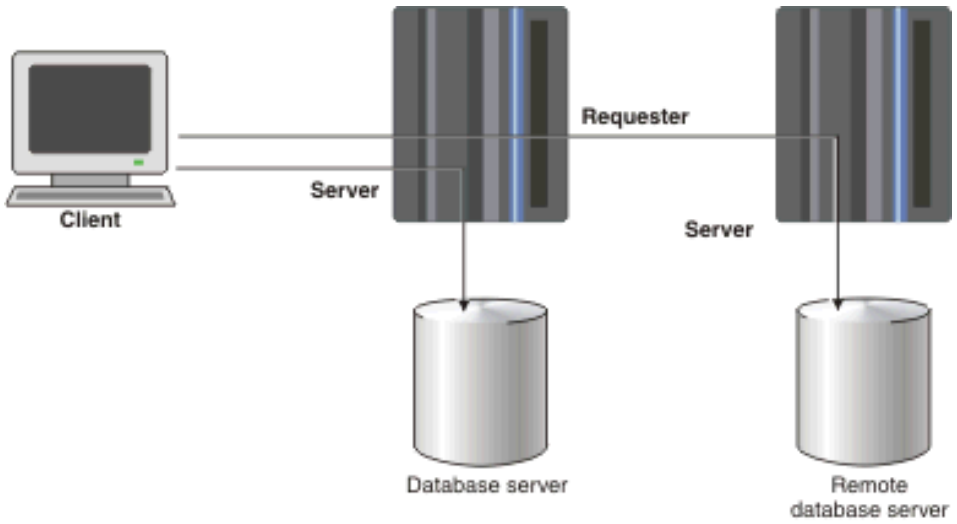


Figure 30. Client/server processing environment

Connectivity in distributed environments

Connectivity in the client/server environment enables communication between applications and database systems on disparate operating systems.

Connectivity in the client/server environment requires an architecture that can handle the stringent performance requirements of a transaction-based system and the flexibility of a decision-support system by using ODBC or JDBC.

Using standard communication protocols, Db2 can bind and rebind packages at other servers and run the statements in those packages. Communication protocols are rules for managing the flow of data across a computer network just as traffic lights and traffic rules manage the flow of car traffic. These protocols are invisible to Db2 applications.

In a distributed environment, applications can connect to multiple databases on different servers and can complete transactions, including commit and rollback operations, at the same time. This type of connectivity is known as a *distributed unit of work*.

Distributed data access

Distributed computing environments typically involve requests from users at one DBMS client that are processed by a DBMS server. The server DBMS is typically remote to the client. Certain programming techniques and performance implications apply to distributed computing environments.

The Db2 distributed environment supports both a two-tier and a multitier architecture.

A DBMS, whether local or remote, is known to your Db2 subsystem by its location name. Remote systems use the location name, or an alias location name, to access a Db2 subsystem. You can define a maximum of eight location names for a Db2 subsystem.

The location name of the Db2 subsystem is defined in the BSDS during Db2 installation. The communications database (CDB) records the location name and the network address of a remote DBMS. The CDB is a set of tables in the Db2 catalog.

The primary method that Db2 uses for accessing data at a remote server is based on Distributed Relational Database Architecture (DRDA).

If your application performs updates to two or more DBMSs, a transaction manager coordinates the two-phase commit process and guarantees that units of work are consistently committed or rolled back. IfDb2 requests updates to two or more DBMSs, Db2 acts as the transaction manager. The distributed commit protocols that are used on the network connection dictate whether both DBMSs can perform updates or whether updates are restricted to a single DBMS.

The examples that follow show statements that you can use to access distributed data.

GUPI

Example 1: To access data at a remote server, write statements like the following example:

```
EXEC SQL
  CONNECT TO CHICAGO;
SELECT * FROM IDEMP01.EMP
  WHERE EMPNO = '000030';
```

You can also accomplish the same task by writing the query like the following example:

```
SELECT * FROM CHICAGO.IDEMP01.EMP
  WHERE EMPNO = '000030';
```

Before you can execute either query at location CHICAGO, you must bind a package at the CHICAGO server.

Example 2: You can call a stored procedure to access data at a remote server. Your program executes these statements:

```
EXEC SQL
  CONNECT TO ATLANTA;
EXEC SQL
  CALL procedure_name (parameter_list);
```

The parameter list is a list of host variables that is passed to the stored procedure and into which it returns the results of its execution. The stored procedure must exist at location ATLANTA.

GUPI

Related concepts

Distributed data
Distributed data is data that resides on a DBMS other than your local system.

Data distribution and Web access
The ability to distribute data and provide Web access to that data is vital to service providers and their customers.

Effects of distributed data on program preparation
In a distributed data environment, several items affect precompile and bind options that are used for DRDA access and package resolution.

Web application environment
Web-based applications run on a web application server and access data on an enterprise information system, such as a Db2 database server. The components of web-based applications are spread across multiple tiers, or layers.

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Ways to implement distributed data in programs

You can connect to a remote server in different ways. You can code an application that uses DRDA to access data at a remote location by using either CONNECT statements or three-part names and aliases.

Using either method, you must bind the DBRMs for the SQL statements that are to execute at the server to packages that reside at the server.

- At the local Db2 subsystem, use the BIND PLAN command to build an application plan.
- At the remote location, use the BIND PACKAGE command to build an application package that uses the local application plan.

Related concepts

The DRDA database protocol (Db2 Installation and Migration)

Explicit CONNECT statements

Using CONNECT statements provides application portability across all Db2 clients and requires the application to manage connections.

With the CONNECT statement, an application program explicitly connects to each server. You must bind the DBRMs for the SQL statements that are to execute at the server to packages that reside at that server.

The application connects to each server based on the location name in the CONNECT statement. You can explicitly specify a location name, or you can specify a location name in a host variable. Issuing the CONNECT statement changes the special register CURRENT SERVER to show the location of the new server.

GUPI

Example: Assume that an application includes a program loop that reads a location name, connects to the location, and executes an INSERT statement. The application inserts a new location name into a host variable, LOCATION_NAME, and executes the following statements:

```
EXEC SQL
   CONNECT TO :LOCATION_NAME;
EXEC SQL
   INSERT INTO IDP101.PROJ VALUES (:PROJNO, :PROJNAME, :DEPTNO,
   :RESPEMP, :MAJPROJ);
```

The host variables match the declaration for the PROJ table.

GUPI

Db2 guarantees the consistency of data across a distributed transaction. To keep the data consistent at all locations, the application commits the work only after the program loop executes for all locations. Either every location commits the INSERT, or, if a failure prevents any location from inserting, all other locations roll back the INSERT.

Related concepts

The DRDA database protocol (Db2 Installation and Migration)

Three-part names and aliases in distributed data applications

Using three-part object names and aliases provides the application with location transparency; objects can move to a new location without requiring changes to the application. Instead, the DBMS manages the underlying connections.

You can use three-part names to access data at a remote location, including tables and views. Using a three-part name, or an alias, an application program implicitly connects to each server. With these access methods, the database server controls where the statement executes.

A three-part name consists of:

- A LOCATION name that uniquely identifies the remote server that you want to access
- An AUTHORIZATION ID that identifies the owner of the object (the table or view) at the location that you want to access
- An OBJECT name that identifies the object at the location that you want to access

GUPI

Example: This example shows how an application uses a three-part name in INSERT, PREPARE, and EXECUTE statements. Assume that the application obtains a location name, 'SAN_JOSE'. Next, it creates the following character string:

```
INSERT INTO SAN_JOSE.IDP101.PROJ VALUES (?,?,?,?,?)
```

The application assigns the character string to the variable INSERTX, and then executes these statements:

```
EXEC SQL
   PREPARE STMT1 FROM :INSERTX;
EXEC SQL
   EXECUTE STMT1 USING :PROJNO, :PROJNAME, :DEPTNO, :RESPEMP, :MAJPROJ;
```

The host variables match the declaration for the PROJ table.

GUPI

Recommendation: If you plan to port your application from a z/OS server to another server, you should not use three-part names. For example, a client application might connect to a z/OS server and then issue a three part-name for an object that resides on a Linux server. Db2 for z/OS is the only server that automatically forwards SQL requests that reference objects that do not reside on the connected server.

A convenient alternative approach is to use aliases when creating character strings that become prepared statements, instead of using full three-part names.

Deciding between three-part names and aliases

Three-part names and aliases each have different advantages.

You can always use three-part names to reference data at another remote server. The advantage of three-part names is that they allow application code to run at different Db2 locations without the additional overhead of maintaining aliases. However, if the table locations change, you must also change the affected applications.

The advantage of aliases is that they allow you to move data around without needing to modify application code or interactive queries. However, if you move a table or view, you must drop the aliases that refer to those tables or views. Then, you can re-create the aliases with the new location names.

Related concepts

Aliases
An *alias* is a substitute for the three-part name of a table or view.

The DRDA database protocol (Db2 Installation and Migration)

Ways that other tasks are affected by distributed data

When you operate in a distributed environment, you need to consider how the environment affects planning and programming activities.

Effects of distributed data on planning

When you work in a distributed environment, you need to consider how authorization works and the cost of running SQL statements.

The appropriate authorization ID must have authorization at a remote server to connect to and to use resources at that server.

You can use the *resource limit facility* at the server to set resource limits for distributed SQL statements. Resource limits apply only to certain dynamic SQL statements. Using this facility, a server can govern how much of its resources a given package can consume by using DRDA access.

Related concepts

Distributed data
Distributed data is data that resides on a DBMS other than your local system.

Remote servers and distributed data (Db2 Application programming and SQL)
Coding methods for distributed data (Db2 Application programming and SQL)

Related tasks

Granting privileges for accessing distributed data (Managing Security)
Improving performance for applications that access distributed data (Db2 Performance)

Limiting resources for statements from remote locations (Db2 Performance)

Related information

Managing distributed access (Managing Security)

Effects of distributed data on programming

There are several effects that you must be aware of when you write programs for a distributed environment.

Keep in mind the following considerations when you write programs that are used in a distributed environment:

- Stored procedures

  If you use DRDA access, your program can call stored procedures. Stored procedures behave like subroutines that can contain SQL statements and other operations.

- Three-part names and multiple servers

  Assume that a statement runs at a remote server (server 1). That statement uses a three-part name or an alias that resolves to a three-part name. The statement includes a location name of a different server (server 2). To ensure that access to the second remote server is by DRDA access, bind the package that contains the three-part name at the second server.

- SQL differences at servers other than Db2 for z/OS

  With explicit connections, a program that uses DRDA access can use SQL statements that a remote server supports, even if the local server does not support them. A program that uses three-part object names cannot execute non-z/OS SQL.

Effects of distributed data on program preparation

In a distributed data environment, several items affect precompile and bind options that are used for DRDA access and package resolution.

The following table lists the z/OS precompiler options that are relevant to preparing a package that is to be run using DRDA access.

Table 20. Db2 precompiler options for DRDA access

z/OS precompiler options	Usage
CONNECT	Use CONNECT(2) to allow your application program to make updates at more than one DBMS.
SQL	Use SQL(ALL) for binding a package to a non-z/OS server; otherwise, use SQL(Db2).

Usually, binding a package to run at a remote location is like binding a package to run at your local Db2 subsystem. Binding a plan to run the package is like binding any other plan. The following table gives you guidelines for which z/OS bind options to choose when binding a package and planning to run using DRDA access.

Table 21. z/OS bind options for DRDA access

z/OS bind options	Usage
DEFER(PREPARE)	For dynamic SQL, use DEFER(PREPARE) to send PREPARE and EXECUTE statements together over the network to improve performance.
SQLERROR	Use SQLERROR(CONTINUE) to create a package even if the bind process finds SQL errors.
SQLRULES	Use SQLRULES(Db2) for more flexibility in coding your applications, particularly for LOB data, and to improve performance.

JDBC, SQLJ, and ODBC use different methods for binding packages that involve less preparation for accessing a z/OS server.

The CURRENT PACKAGE PATH special register provides a benefit for applications that use DRDA from a z/OS requester. The package collection ID is resolved at the server. Applications on the server can take advantage of the list of collections, and network traffic is minimal.

GUPI

Example: Assume that five packages exist and that you want to invoke the first package at the server. The package names are SCHEMA1.PKG1, SCHEMA2.PKG2, SCHEMA3.PKG3, SCHEMA4.PKG4, and SCHEMA5.PKG5. Rather than issuing a SET CURRENT PACKAGESET statement to invoke each package, you can use a single SET statement if the server supports the CURRENT PACKAGE PATH special register:

```
SET CURRENT PACKAGE PATH = SCHEMA1, SCHEMA2, SCHEMA3, SCHEMA4, SCHEMA5;
```

GUPI

Related concepts

Preparation process for an application program
How you prepare an application program to run depends on the type of application. The program preparation steps for applications vary based on the type of programming language that is used.

How updates are coordinated across distributed systems
Various products are available to work with Db2 to coordinate updates across a distributed transaction. Db2 coordinates updates at servers that support different types of connections.
Related concepts
The DRDA database protocol (Db2 Installation and Migration)

Db2 transaction manager support

Db2 supports a wide range of transaction manager products to coordinate updates across a distributed transaction. A distributed transaction typically involves multiple recoverable resources, such as Db2 tables, MQSeries® messages, and IMS databases.

Application environments that use Db2 Connect to access Db2 remotely can use the following transaction manager products:

- WebSphere Application Server
- CICS
- IBM TXSeries® (CICS and Encina)
- WebSphere MQ
- Microsoft Transaction Server (MTS)
- Java applications that support Java Transaction API (JTA) and Enterprise JavaBeans (EJBs)
- BEA (Tuxedo and WebLogic)
- Other transaction manager products that support standard XA protocols

  The XA interface is a bidirectional interface between a transaction manager and resource managers that provides coordinated updates across a distributed transaction. The Open Group defined XA protocols based on the specification Distributed TP: The XA Specification.

Application environments that access Db2 locally can use the following transaction manager products:

- WebSphere Application Server
- CICS transaction server
- IMS

For application environments that do not use a transaction manager, Db2 coordinates updates across a distributed transaction by using DRDA-protected connections.

Related concepts

The DRDA database protocol (Db2 Installation and Migration)

Servers that support two-phase commit

Updates in a two-phase commit situation are coordinated if they must all commit or all roll back in the same unit of work.

Example: You can update an IMS database and a Db2 table in the same unit of work. Suppose that a system or communication failure occurs between committing the work on IMS and on Db2. In that case, the two programs restore the two systems to a consistent point when activity resumes.

Db2 coordinates commits even when a connection is using one-phase commit in a distributed transaction. In this case, however, only one location can perform an update.

Related concepts

Coordinated updates for consistency between servers

In a distributed system, a transaction might occur at more than one server. To ensure data consistency, each subsystem that participates in a single transaction must coordinate update operations. Transactions must be either committed or backed out.

The DRDA database protocol (Db2 Installation and Migration)

Servers that do not support two-phase commit

In a distributed transaction, Db2 can coordinate a mixture of two-phase and one-phase connections.

You cannot have coordinated updates with a DBMS that does not implement two-phase commit. You can, however, achieve the effect of coordinated updates when you access a server that does not implement two-phase commit; such a server is called a *restricted system*.

Db2 prevents you from updating both a restricted system and any other system in the same unit of work. In this context, *update* includes the statements INSERT, DELETE, UPDATE, CREATE, ALTER, DROP, GRANT, REVOKE, and RENAME.

You can achieve the effect of coordinated updates with a restricted system. You must first update one system and commit that work, and then update the second system and commit its work. However, suppose that a failure occurs after the first update is committed and before the second update is committed. No automatic provision is available to bring the two systems back to a consistent point. Your program must handle that task.

When you access a mixture of systems, some of which might be restricted, you can take the following actions to ensure data integrity:

- Read from any of the systems at any time.
- Update any one system many times in one unit of work.
- Update many systems, including CICS or IMS, in one unit of work if no system is a restricted system. If the first system you update is not restricted, any attempt to update a restricted system within a unit of work results in an error.
- Update one restricted system in a unit of work. You can do this action only if you do not try to update any other system in the same unit of work. If the first system that you update is restricted, any attempt to update another system within that unit of work results in an error.

Related concepts

The DRDA database protocol (Db2 Installation and Migration)

Ways to reduce network traffic

The key to improving performance in a network computing environment is to minimize network traffic.

Stored procedures are an excellent method for sending many SQL statements in a single network message and, as a result, running many SQL statements at the Db2 server. This topic introduces you to other ways to improve performance when accessing remote servers.

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Improvements in query efficiency

Queries almost always execute faster on a local server than they do when the same query is sent to a remote server. To increase efficiency when accessing remote servers, try to write queries that send few messages over the network.

For example:

- Reduce the number of columns and rows in the result table that is returned to your application. Keep your SELECT lists as short as possible. Creative use of the clauses WHERE, GROUP BY, and HAVING can eliminate unwanted data at the remote server.
- Use FOR READ ONLY. For example, retrieving thousands of rows as a continuous stream is reasonable. Sending a separate message for each one can be much slower.
- When possible, do not bind application plans and packages with ISOLATION(RR). If your application does not need to refer again to rows it reads once, another isolation level might reduce lock contention and message overhead during COMMIT processing.
- Minimize the use of parameter markers.

  When your program uses DRDA access, Db2 can streamline the processing of dynamic queries that do not have parameter markers. However, parameter markers are needed for effective dynamic statement caching.

  When a Db2 requester encounters a PREPARE statement for such a query, it anticipates that the application is going to open a cursor. The requester therefore sends the server a single message that contains a combined request for PREPARE, DESCRIBE, and OPEN. A Db2 server that receives this message sequence returns a single reply message sequence that includes the output from the PREPARE, DESCRIBE, and OPEN operations. As a result, the number of network messages that are sent and received for these operations is reduced from two to one.

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Choosing an ISOLATION option (Db2 Performance)

Related reference

group-by-clause (Db2 SQL)

having-clause (Db2 SQL)

read-only-clause (Db2 SQL)

where-clause (Db2 SQL)

PREPARE (Db2 SQL)

Reduction in the volume of messages

Db2 capabilities that combine multiple rows of data during fetch and insert operations can significantly reduce the number of messages that are sent across the network. Those capabilities include block fetch and rowset fetches and inserts.

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Block fetch

You can use block fetch to retrieve a set of rows and transmit them all in one message over the network.

Db2 uses a *block fetch* to group the rows that an SQL query retrieves into as large a "block" of rows as can fit in a message buffer, and then transmits the block over the network. By sending multiple rows in a block, Db2 avoids sending a message for every row.

A block fetch is used only with cursors that do not update data.

Db2 can use two different types of block fetch:

Limited block fetch
An operation that optimizes data transfer by minimizing the number of messages that are transmitted from the requester whenever a remote fetch operation is performed.

Continuous block fetch
An operation that optimizes data transfer by minimizing the number of messages that are transmitted from the requester to retrieve the entire result set. In addition, overlapped processing is performed at the requester and the server.

To use block fetch, Db2 must determine that the cursor is not used for update or delete. You can indicate in your program by adding the clause FOR READ ONLY or FOR FETCH ONLY to the query. If you do not specify FOR READ ONLY or FOR FETCH ONLY, the way in which Db2 uses the cursor determines whether it uses block fetch. For scrollable cursors, the sensitivity of the cursor and the bind options affect whether Db2 can use block fetch.

Related concepts
Limited block fetch (Db2 Performance)
Continuous block fetch (Db2 Performance)
Related tasks
Improving performance for applications that access distributed data (Db2 Performance)
Enabling block fetch for distributed applications (Db2 Performance)
Related reference
read-only-clause (Db2 SQL)
DECLARE CURSOR (Db2 SQL)

Rowset fetches and inserts
For rowset-positioned cursors, when the cursor is opened for rowset processing, the answer set is returned in a single query block. The query block contains exactly the number of rows that are specified for the rowset.

Because a rowset is returned in a single query block, the size of a rowset is limited to 10 MB. This rowset size minimizes the impact to the network when retrieving a large rowset with a single fetch operation.

Rowset-positioned cursors also allow multiple-row inserts. The INSERT statement, in addition to the FOR *n* ROWS clause, inserts multiple rows into a table or view, by using values that host-variable arrays provide. With multiple-row inserts, rather than INSERT statements being sent for each individual insert, all insert data is sent in a single network message.

Related concepts
Row retrieval with a cursor
Db2 has a mechanism called a *cursor*. Using a cursor is like keeping your finger on a particular line of text on a printed page.

Related tasks
Improving performance for applications that access distributed data (Db2 Performance)

Optimization for large and small result sets
Several options on the SELECT statement let you limit the number of rows that are returned to a client program.

Enabling a Db2 client to request that multiple query blocks on each transmission can reduce network activity and improve performance significantly for applications that use DRDA access to download large amounts of data.

You can specify a large value of *n* in the OPTIMIZE FOR *n* ROWS clause of a SELECT statement to increase the number of DRDA query blocks that a Db2 server returns in each network transmission for a nonscrollable cursor.

If *n* is greater than the number of rows that fit in a single DRDA query block, the OPTIMIZE FOR *n* ROWS clause lets the DRDA client request multiple blocks of query data on each network transmission instead of requesting another block when the first block is full. This use of the OPTIMIZE FOR *n* ROWS clause is

intended for applications that open a cursor and download large amounts of data. The OPTIMIZE FOR *n* ROWS clause does not affect scrollable cursors.

When a client does not need all the rows from a potentially large result set, preventing the Db2 server from returning all the rows for a query can reduce network activity and improve performance significantly for DRDA applications. You can use either the OPTIMIZE FOR *n* ROWS clause or the FETCH FIRST *n* ROWS ONLY clause of a SELECT statement to limit the number of rows that are returned to a client program.

Related tasks

Improving performance for applications that access distributed data (Db2 Performance)

Optimizing retrieval for a small set of rows (Db2 Application programming and SQL)

Fetching a limited number of rows (Db2 Performance)

Related reference

optimize-clause (Db2 SQL)

fetch-first-clause (Db2 SQL)

Support for high availability

Because Db2 provides support for high availability, frequently starting or stopping Db2 is not necessary.

You can run Db2 for several weeks without stopping and starting the subsystem. Some customers have managed to keep Db2 running continuously for several years. One key to achieving high availability is to use data sharing. Data sharing allows access to data even when one Db2 subsystem in a group is stopped. Another key to high availability is the ability to get the Db2 subsystem back up and running quickly after an unplanned outage.

- Some restart processing can occur concurrently with new work. Also, you can choose to postpone some processing.
- During a restart, Db2 applies data changes from the log. This technique ensures that data changes are not lost, even if some data was not written at the time of the failure. Some of the process of applying log changes can run in parallel
- You can register Db2 to the Automatic Restart Manager of z/OS. This facility automatically restarts Db2 if it goes down as a result of a failure.

Related concepts

Backup, recovery, and restart

Although high availability of data is a goal for all Db2 subsystems, unplanned outages are difficult to avoid entirely. A good backup, recovery, and restart strategy, however, can reduce the elapsed time of an unplanned outage.

Db2 data sharing

In Db2 *data sharing*, applications that reside in multiple Db2 subsystems in a Parallel Sysplex can read from and write to the same Db2 for z/OS data concurrently, with integrity, performance, scalability, and dynamic workload balancing.

Db2 subsystems that share data must belong to a Db2 data sharing group, which runs on a IBM Z Parallel Sysplex cluster. A *data sharing group* is a collection of one or more Db2 subsystems that access shared Db2 data.

A Parallel Sysplex is a cluster of z/OS systems that communicate and cooperate with each other. The Parallel Sysplex is a highly sophisticated cluster architecture. It consists of two key pieces of technology:

Coupling facility
: Provides specialized hardware, specialized high-speed links and adapters, and a shared, nonvolatile electronic storage for fast intersystem data sharing protocols.

Sysplex Timer
: Provides a common time source across all the systems in the cluster, thereby delivering an efficient way to provide log-record sequencing and event ordering across the different systems.

The coupling facility and the Sysplex Timer are exclusive to the IBM Z environment. They provide strong performance and scalability in a multisystem clustered DBMS environment with shared disks.

Each Db2 subsystem that belongs to a particular data sharing group is a *member* of that group. All members of a data sharing group use the same shared Db2 catalog.

You can use some capabilities that are described in this information regardless of whether you share data. The term *data sharing environment* refers to a situation in which a data sharing group is defined with at least one member. In a non-data-sharing environment, no group is defined.

Related concepts

Introduction to Db2 data sharing (Db2 Data Sharing Planning and Administration)

Planning for Db2 data sharing (Db2 Installation and Migration)

Availability and scalability for large businesses

Large businesses choose Db2 for z/OS because they need a robust database server that ensures superior availability and scalability.

Related tasks

Installing, migrating, and enabling Db2 data sharing (Db2 Installation and Migration)

Advantages of Db2 data sharing

You can use data sharing to enhance the capabilities of Db2.

Db2 data sharing improves Db2 availability, enables scalable growth, and provides more flexible ways to configure your environment. You don't need to change SQL in your applications to use data sharing, although you might need to do some tuning for optimal performance.

Related concepts

Advantages of Db2 data sharing (Db2 Data Sharing Planning and Administration)

Improved availability of data

Db2 data sharing helps you meet your service objective by improving availability during both planned and unplanned outages.

As the following figure illustrates, if one subsystem fails, users can access their Db2 data from another subsystem. Transaction managers are informed that Db2 is down and can switch new user work to another Db2 subsystem in the group. For unplanned outages, the z/OS automatic restart manager can automate restart and recovery.

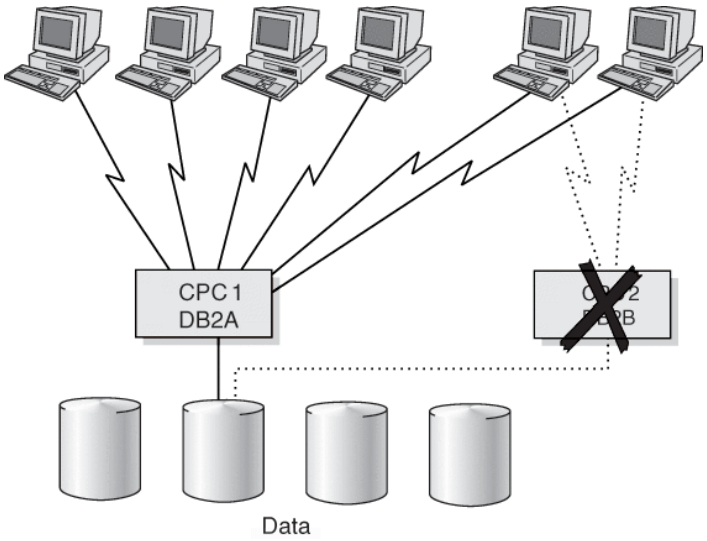


Figure 31. How data sharing improves availability during outages

Although the increased availability of Db2 has some performance cost, the overhead for interprocessor communication and caching changed data is minimized. Db2 provides efficient locking and caching

mechanisms and uses coupling facility hardware. A *coupling facility* is a special logical partition that runs the coupling facility control program. It provides high-speed caching, list processing, and locking functions in a Sysplex. The Db2 structures in the coupling facility benefit from high availability. The coupling facility uses automatic structure rebuild and duplexing of the structures that are used for caching data.

Related concepts

Advantages of Db2 data sharing (Db2 Data Sharing Planning and Administration)

Scalable growth

As you move more data processing into a Db2 environment, your processing needs can exceed the capacity of a single system. Data sharing might relieve that constraint.

Db2 for z/OS is optimized to use your existing hardware more efficiently and manage a larger workload. Db2 now has more 64 bit storage, and supports more concurrent threads than previous versions. These improvements greatly increase the capabilities of a single system, which reduces the cost of new hardware, software, and maintenance. However, there are several ways to scale the capabilities of your system to meet your business needs.

Without data sharing

Without Db2 data sharing, you have the following options:

- Copy the data, or split the data into separate Db2 subsystems.

  This approach requires that you maintain separate copies of the data. No communication takes place among Db2 subsystems, and the Db2 catalog is not shared.
- Install another Db2 subsystem, and rewrite applications to access the original data as distributed data.

  This approach might relieve the workload on the original Db2 subsystem, but it requires changes to your applications and has performance overhead of its own. Nevertheless, for Db2 subsystems that are separated by great distance or for a Db2 subsystem that needs to share data with a system that outside the data sharing group, the distributed data facility is still your only option.
- Install a larger processor and move data and applications to that machine.

  This option can be expensive. In addition, this approach requires your system to come down while you move to the new, larger machine.

With data sharing

With Db2 data sharing, you can take advantage of the following benefits:

Incremental growth

The Parallel Sysplex cluster can grow incrementally. You can add a new Db2 subsystem onto another central processor complex and access the same data through the new Db2 subsystem. You no longer need to manage copies or distribute data. All Db2 subsystems in the data sharing group have concurrent read-write access, and all Db2 subsystems use a single Db2 catalog.

Workload balancing

Db2 data sharing provides flexibility for growth and workload balancing. With the partitioned data approach to parallelism (sometimes called the *shared-nothing* architecture), a one-to-one relationship exists between a particular DBMS and a segment of data. By contrast, data in a Db2 data sharing environment does not need to be redistributed when you add a new subsystem or when the workload becomes unbalanced. The new Db2 member has the same direct access to the data as all other existing members of the data sharing group.

Db2 works closely with the z/OS Workload Manager (WLM) to ensure that incoming work is optimally balanced across the systems in the cluster. WLM manages workloads that share system resources and have different priorities and resource-use characteristics.

Example: Assume that large queries with a low priority are running on the same system as online transactions with a higher priority. WLM can ensure that the queries do not monopolize resources and

do not prevent the online transactions from achieving acceptable response times. WLM works in both a single-system and a multisystem (data sharing) environment.

Capacity when you need it
A data sharing configuration can handle your peak loads. You can start data sharing members to handle peak loads, such as end-of-quarter processing, and then stop them when the peak passes.

You can take advantage of all these benefits, whether your workloads are for online transaction processing (OLTP), or a mixture of OLTP, batch, and queries.

Higher transaction rates

Data sharing gives you opportunities to put more work through the system. As the following figure illustrates, you can run the same application on more than one Db2 subsystem to achieve transaction rates that are higher than are possible on a single subsystem.

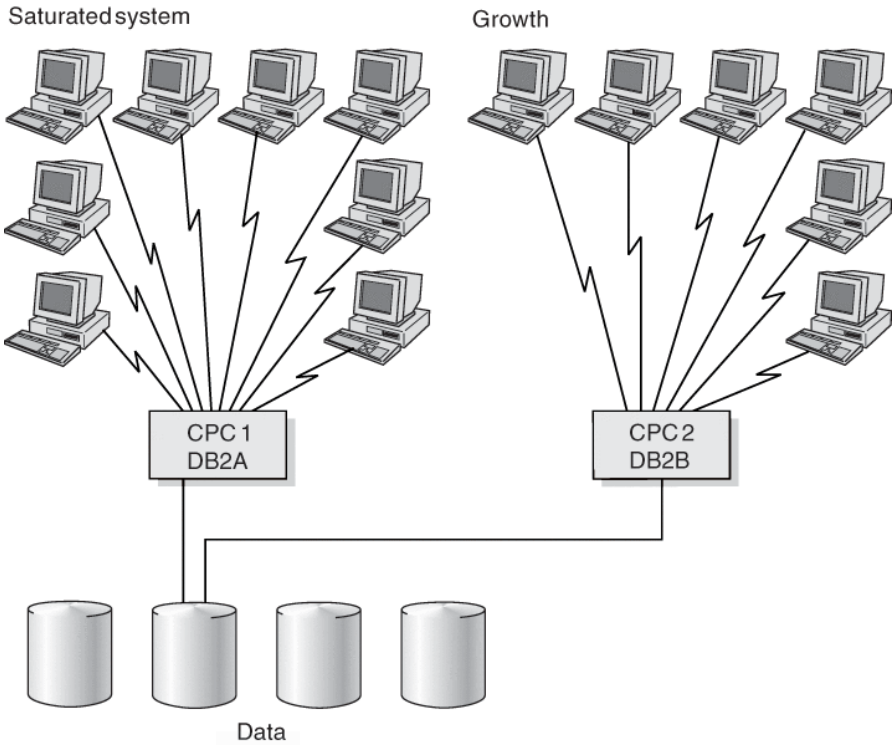


Figure 32. How data sharing enables growth

Related concepts
Advantages of Db2 data sharing (Db2 Data Sharing Planning and Administration)

Flexible configurations
When you use Db2 data sharing, you can configure your system environment much more flexibly.

As the following figure shows, you can have more than one Db2 data sharing group on the same z/OS Sysplex. You might, for example, want one group for testing and another for production data.

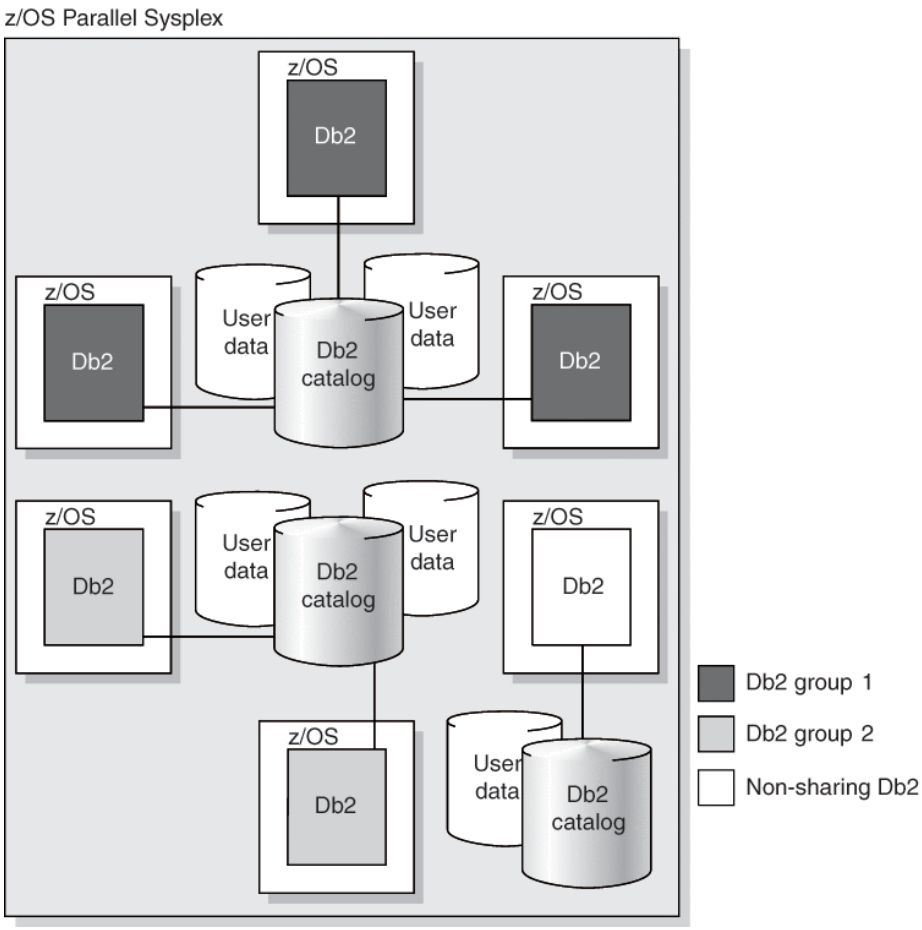


Figure 33. A possible configuration of Db2 data sharing groups

You can also run multiple members on the same z/OS image (not shown in this figure).

Flexible operational systems

The following figure shows how, with data sharing, you can have query user groups and online transaction user groups on separate z/OS images. This configuration lets you tailor each system specifically for that user set, control storage contention, and provide predictable levels of service for that set of users. Previously, you might have needed to manage separate copies of data to meet the needs of different user groups.

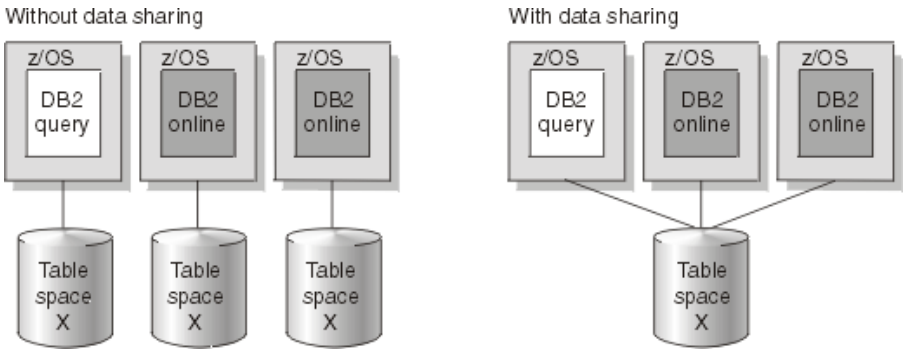


Figure 34. Flexible configurations with Db2 data sharing

Flexible decision support systems

The following figure shows two different decision support configurations. A *typical configuration* separates the operational data from the decision support data. Use this configuration when the

operational system has environmental requirements that are different from those of the decision support system. The decision support system might be in a different geographical area, or security requirements might be different for the two systems.

Db2 offers another option—a *combination configuration*. A combination configuration combines your operational and decision support systems into a single data sharing group and offers these advantages:

- You can occasionally join decision support data and operational data by using SQL.
- You can reconfigure the system dynamically to handle fluctuating workloads. For example, you might choose to dedicate CPCs to decision support processing or operational processing at different times of the day or year.
- You can reduce the cost of computing:
  - The infrastructure that is used for data management is already in place.
  - You can create a prototype of a decision support system in your existing system and then add processing capacity as the system grows.

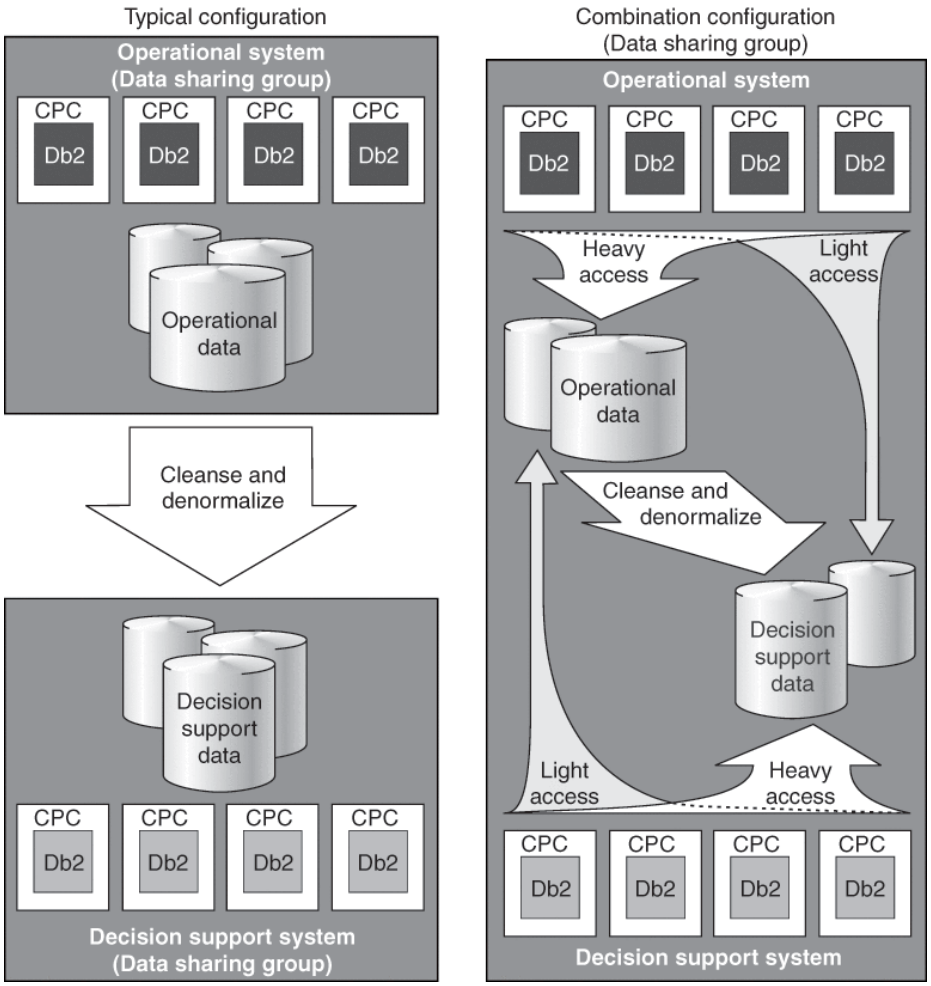


Figure 35. Flexible configurations for decision support

To set up a combination configuration, separate decision support data from operational data as much as possible. Buffer pools, disks, and control units that you use in your decision support system must be separate from those that you use in your operational system. This separation greatly minimizes any negative performance impact on the operational system.

If you are unable to maintain that level of separation or if you have separated your operational data for other reasons such as security, using a separate decision support system is your best option.

Flexibility to manage shared data

Data sharing can simplify the management of applications that must share some set of data, such as a common customer table. Maybe these applications were split in the past for capacity or availability reasons. But with the split architecture, the shared data must be kept in synch across the multiple systems (that is, by replicating data).

Data sharing gives you the flexibility to configure these applications into a single Db2 data sharing group and to maintain a single copy of the shared data that can be read and updated by multiple systems with good performance. This option is especially powerful when businesses undergo mergers or acquisitions or when data centers are consolidated.

Related concepts

Advantages of Db2 data sharing (Db2 Data Sharing Planning and Administration)

Protected investments in people and skills

Your investment in people and skills is protected when you use Db2 data sharing because existing SQL interfaces and attachments remain intact when sharing data.

You can bind a package or plan on one Db2 subsystem and run that package or plan on any other Db2 subsystem in a data sharing group.

Related concepts

Advantages of Db2 data sharing (Db2 Data Sharing Planning and Administration)

How Db2 protects data consistency in a data sharing environment

Applications can access data from any Db2 subsystem in a data sharing group. Many subsystems can potentially read and write the same data. Db2 uses special data sharing mechanisms for locking and caching to ensure data consistency.

When multiple members of a data sharing group have opened the same table space, index space, or partition, and at least one of them has opened it for writing, the data is said to be of inter-Db2 read-write interest to the members. (Sometimes this information uses the term "inter-Db2 interest.") To control access to data that is of inter-Db2 interest, whenever the data is changed, Db2 caches it in a storage area that is called a *group buffer pool (GBP)*.

Db2 dynamically detects inter-Db2 interest, which means that Db2 can invoke intersystem data sharing protocols only when data is actively read-write shared between members. Db2 can detect when data is not actively intersystem read-write shared. In these cases, data sharing locking or caching protocols are not needed, which can result in better performance.

When inter-Db2 read-write interest exists in a particular table space, index, or partition, this inter-Db2 read-write interest is *dependent* on the group buffer pool, or *group buffer pool dependent*.

You define group buffer pools by using coupling facility resource management (CFRM) policies.

The following figure shows the mapping that exists between a group buffer pool and the buffer pools of the group members. For example, each Db2 subsystem has a buffer pool named BP0. For data sharing, you must define a group buffer pool (GBP0) in the coupling facility that maps to buffer pool BP0. GBP0 is used for caching the Db2 catalog table space and its index, and any other table spaces, indexes, or partitions that use buffer pool BP0.

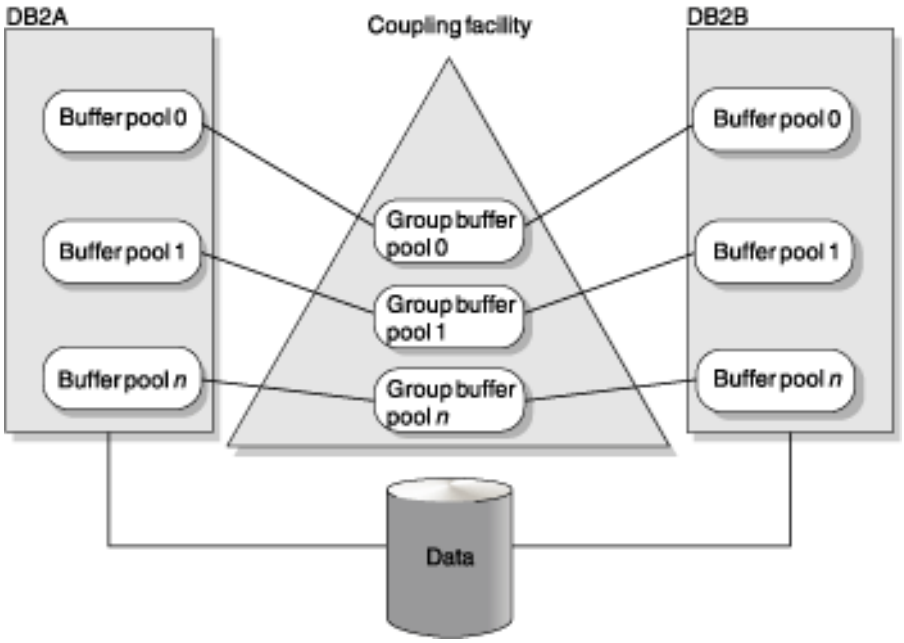


Figure 36. Relationship of buffer pools to group buffer pools

The same group buffer pool cannot reside in more than one coupling facility (unless it is duplexed).

When a particular page of data is changed, Db2 caches that page in the group buffer pool. The coupling facility invalidates any image of the page that might exist in the buffer pools that are associated with each member. Then, when another Db2 subsystem subsequently requests that same data, that Db2 subsystem looks for the data in the group buffer pool.

Performance benefits

The coupling facility provides fast, global locking operations for concurrency control. The Parallel Sysplex offers the following performance and scalability benefits:

- Changed pages are written synchronously to the coupling facility, without the process switching that is associated with disk I/O.
- Buffer invalidation signals are sent and processed without causing any processor interrupts, unlike message-passing techniques.
- A fast hardware instruction detects invalidated buffers, and the coupling facility can refresh invalidated buffers synchronously with no process switching overhead, unlike disk I/O.

Performance options to fit your application's needs

Although the default behavior is to cache only the updated data, you also have options of caching all or none of your data. You even have the option to cache large object (LOB) data.

Related concepts

How Db2 protects data consistency (Db2 Data Sharing Planning and Administration)

How updates are made in a data sharing environment

There are several steps to the update process in a data sharing environment.

You might be interested to know what happens to a page of data as it goes through the update process. The most recent version of the data page is shaded in the illustrations. This scenario also assumes that the group buffer pool is used for caching only the changed data (the default behavior) and that it is *duplexed* for high availability. Duplexing is the ability to write data to two instances of a group buffer pool structure: a *primary group buffer pool* and a *secondary group buffer pool*.

Suppose, that as the following figure shows, an application issues an UPDATE statement from DB2A and that the data does not reside in the member's buffer pool or in the group buffer pool. In this case, DB2A must retrieve the data from disk and update the data in its own buffer pool. Simultaneously, DB2A gets

the appropriate locks to prevent another member from updating the same data at the same time. After the application commits the update, DB2A releases the corresponding locks. The changed data page remains in DB2A's buffer pool. Because no other Db2 subsystem shares the table at this time, Db2 does not use data sharing processing for DB2A's update.

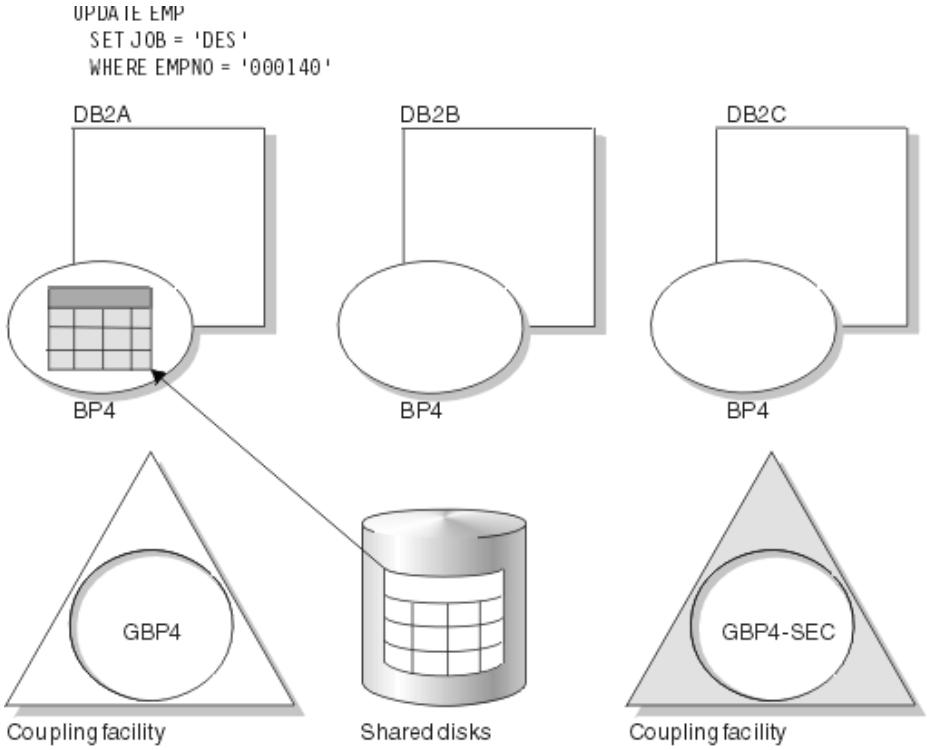


Figure 37. Data is read from disk and updated by an application that runs on DB2A

Next, suppose that another application, which runs on DB2B, needs to update that same data page. (See the following figure.) Db2 knows that inter-Db2 interest exists, so when DB2A commits the transaction, Db2 writes the changed data page to the primary group buffer pool. The write to the backup (secondary) group buffer pool is overlapped with the write to the primary group buffer pool. DB2B then retrieves the data page from the primary group buffer pool.

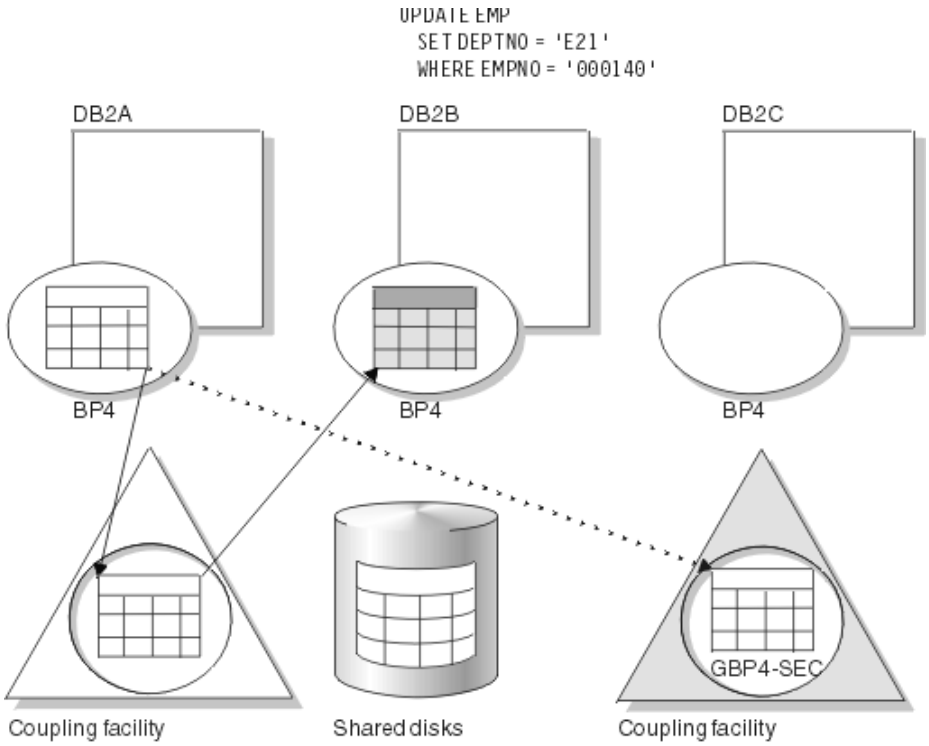


Figure 38. How DB2B updates the same data page

After the application that runs on DB2B commits the update, DB2B moves a copy of the data page into the group buffer pool (both primary and secondary), and the data page is invalidated in DB2A's buffer pool. (See the following figure.) Cross-invalidation occurs from the primary group buffer pool.

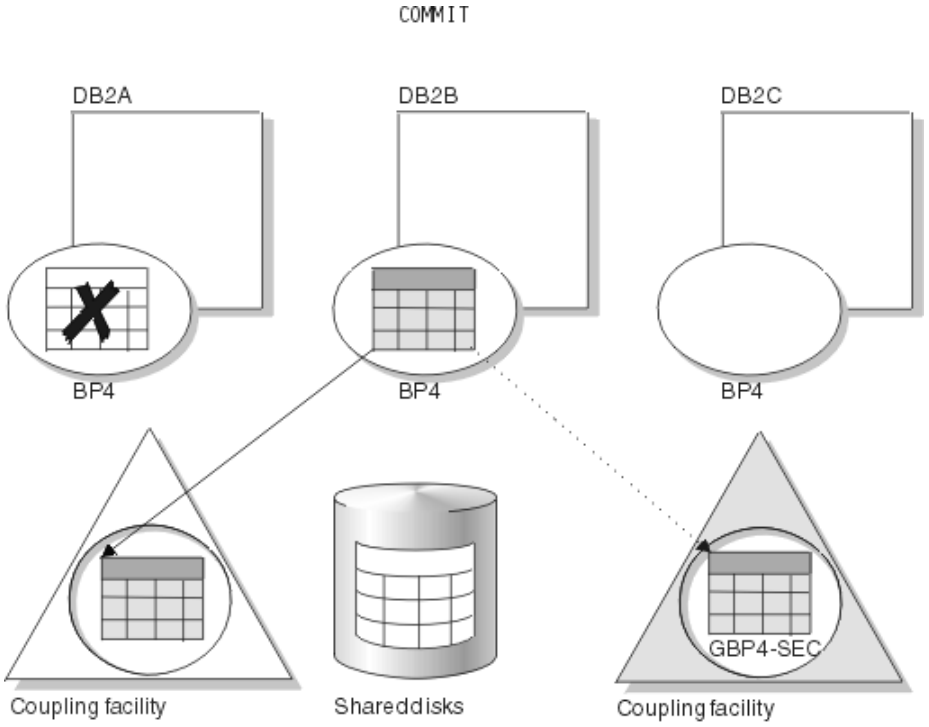


Figure 39. The updated page is written to the group buffer pool

Now, as the following figure shows, when DB2A needs to read the data, the data page in its own buffer pool is not valid. Therefore, it reads the latest copy from the primary group buffer pool.

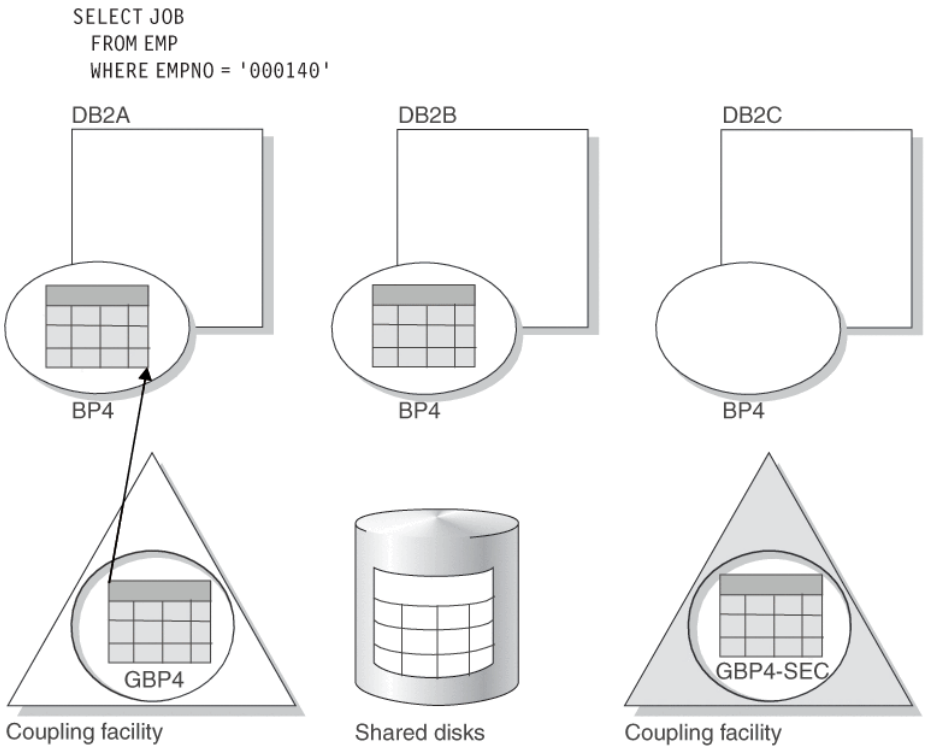


Figure 40. DB2A reads data from the group buffer pool

Unlike disk-sharing systems that use traditional disk I/O and message-passing techniques, the coupling facility offers these advantages:

- The group buffer pool interactions are CPU-synchronous. CPU-synchronous interactions provide good performance by avoiding process-switching overhead and by maintaining good response times.
- The cross-invalidation signals do not cause processor interrupts on the receiving systems; the hardware handles them. The signals avoid process-switching overhead and CPU cache disruptions that can occur if processor interrupts are needed to handle the incoming cross-invalidations.

Related concepts

How an update happens (Db2 Data Sharing Planning and Administration)

How Db2 writes changed data to disk in a data sharing environment

Periodically, Db2 must write changed pages from the group buffer pool to disk. This process is called *castout*. The castout process runs in the background without interfering with transactions.

Suppose that DB2A is responsible for casting out the changed data. That data must first pass through DB2A's address space because no direct connection exists between the coupling facility and disk. (See the following figure.) This data passes through a private buffer, not through the Db2 buffer pools.

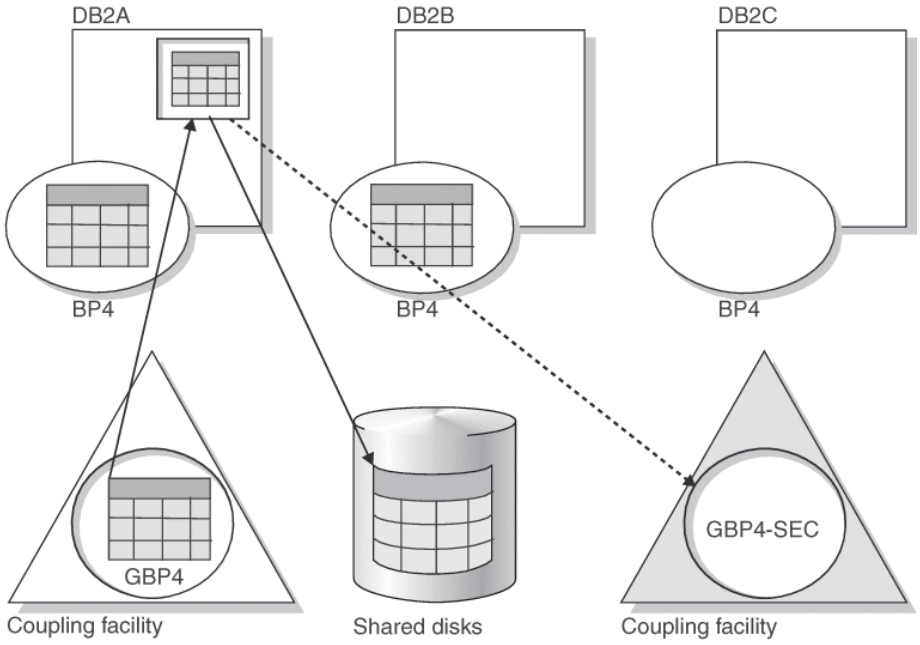


Figure 41. Writing data to disk

When a group buffer pool is duplexed, data is not cast out from the secondary group buffer pool to disk. When a set of pages is written to disk from the primary group buffer pool, Db2 deletes those pages from the secondary group buffer pool.

Related concepts

How Db2 writes changed data to disk (Db2 Data Sharing Planning and Administration)

Ways that other tasks are affected by data sharing

Because data sharing does not change the application interface, application programmers and users have no new tasks. However, some programming, operational, and administrative tasks are unique to the data sharing environment.

The following tasks are unique to the data sharing environment:

- Establishing a naming convention for groups, group members, and resources
- Assigning new members to a data sharing group
- Merging catalog information when data from existing Db2 subsystems moves into a data sharing group

Because the Db2 catalog is shared, data definition, authorization, and control is the same as for non-data-sharing environments. An administrator needs to ensure that every object has a unique name, considering that existing data might be merged into a group. The data needs to reside on shared disks.

Ways that availability is affected by data sharing

Data sharing can provide data availability during an outage, maintain coupling facility availability, and duplex group buffer pools.

Availability during an outage

A significant availability benefit during a planned or unplanned outage of a Db2 group member is that Db2 data remains available through other group members. Some common situations when you might plan for an outage include applying software maintenance, changing a system parameter, or migrating to a new release. For example, during software maintenance, you can apply the maintenance to one member at a time, which leaves other Db2 members available to do work.

Coupling facility availability

When planning your data sharing configuration for the highest availability, you must monitor the physical protection of the coupling facility and the structures within the coupling facility.

For high availability, you must have at least two coupling facilities. One of coupling facility must be physically isolated. The isolated coupling facility must reside in a CPC that does not also contain a Db2 member that is connected to structures in that coupling facility. With at least two coupling facilities, you can avoid a single point of failure.

Duplexing group buffer pools

With more than one coupling facility, you can also consider duplexing the group buffer pools. With duplexing, a secondary group buffer pool is available on standby in another coupling facility, ready to take over if the primary group buffer pool structure fails or if a connectivity failure occurs.

Running some or all of your group buffer pools in duplex mode is one way to achieve high availability for group buffer pools across many types of failures, including lost connections and damaged structures.

pureXML

You can use pureXML with your client applications to manage XML data in Db2 tables. You can store well-formed XML documents in their hierarchical form and retrieve all or portions of those documents.

Because the stored XML data is fully integrated into the Db2 database system, you can access and manage the XML data by using Db2 functions and capabilities.

To efficiently manage traditional SQL data types and XML data, Db2 uses two distinct storage mechanisms. However, the underlying storage mechanism that is used for a given data type is transparent to the application. The application does not need to explicitly specify which storage mechanism to use, or to manage the physical storage for XML and non-XML objects.

XML document storage
The XML column data type is provided for storage of XML data in Db2 tables. Most SQL statements support the XML data type. This enables you to perform many common database operations with XML data, such as creating tables with XML columns, adding XML columns to existing tables, creating indexes over XML columns, creating triggers on tables with XML columns, and inserting, updating, or deleting XML documents.

Alternatively, a decomposition stored procedure is provided so that you can extract data items from an XML document and store those data items in columns of relational tables, using an XML schema that is annotated with instructions on how to store the data items.

XML document retrieval
You can use SQL to retrieve entire documents from XML columns, just as you retrieve data from any other type of column. When you need to retrieve portions of documents, you can specify XPath expressions, through SQL with XML extensions (SQL/XML).

Application development
Application development support of XML enables applications to combine XML and relational data access and storage. The following programming languages support the XML data type:

- Assembler
- C or C++ (embedded SQL or Db2 ODBC)
- COBOL
- Java (pureQuery, JDBC, or SQLJ)
- PL/I
- pureXML

Database administration
Db2 for z/OS database administration support for pureXML includes the following items:

XML schema repository (XSR)

The XML schema repository (XSR) is a repository for all XML schemas that are required to validate and process XML documents that are stored in XML columns or that are decomposed into relational tables.

Utility support

Db2 for z/OS utilities support the XML data type. The storage structure for XML data and indexes is like the storage structure for LOB data and indexes. As with LOB data, XML data is not stored in the base table space, but it is stored in separate table spaces that contain only XML data. The XML table spaces also have their own index spaces. Therefore, the implications of using utilities for manipulating, backing up, and restoring XML data and LOB data are similar.

Performance

Indexing support is available for data stored in XML columns. The use of indexes over XML data can improve the efficiency of queries that you issue against XML documents. An XML index differs from a relational index in that a relational index applies to an entire column, whereas an XML index applies to part of the data in a column. You indicate which parts of an XML column are indexed by specifying an XML pattern, which is a limited XPath expression.

Chapter 3. Management of Db2 operations

Management of a Db2 subsystem on a daily basis requires performing a wide range of tasks. For example, you need to manage authorizations and be prepared to recover from any potential errors or problems.

When you manage a Db2 environment on a daily basis, you need to issue Db2 commands, run Db2 utilities, manage authorizations, and be prepared to recover from potential errors or problems. In addition, you probably want to take advantage of the high availability capabilities that are related to Db2, including the following capabilities:

- You can bind application plans and packages online. By using packages, you can change and rebind smaller units. Using package versions permits binding while the applications continue to run.
- You can define and change databases and authorizations online.
- You can change buffer pool sizes online.
- You can use utilities to reorganize indexes, table spaces, or partitions of indexes or table spaces.
- You can use the data sharing functions of Db2, which enable several Db2 subsystems to process applications on shared data. Although the different subsystems share data, they appear as a single Db2 subsystem to users. Applications can be rerouted to avoid outages if one of the subsystems must be taken down for maintenance.

Several management tools are available to help you easily perform many of the tasks that are associated with daily operations of a Db2 subsystem. For more information, see "Tools for managing Db2 for z/OS" on page 305.

Common ways to interact with Db2 for z/OS

The z/OS operating system is ideal for processing *batch jobs*, workloads that run in the background with little or no human interaction. However, z/OS is as much an interactive operating system as it is a batch processing system.

How can I work interactively with Db2 for z/OS?

You can use the following facilities to work with Db2 for z/OS through direct interaction, such as commands and menu style interfaces:

The z/OS console
You can issue commands to control Db2 from the z/OS console.

Learning resources:

Issuing commands from the z/OS console (Db2 Administration Guide)
-START DB2 (Db2) (Db2 Commands)

Time Sharing Option/Extensions (TSO)
You can use *TSO/E* to log on to z/OS and interactively share the resources of the mainframe. TSO also provides a limited set of basic commands. Using this set is sometimes called using TSO in its *native mode*.

Learning resources:

"TSO attachment facility" on page 294
What is TSO? (z/OS basic skills)
TSO/E Primer

z/OS TSO/E

Interactive System Productivity Facility (ISPF)
ISPF is a menu-driven interface for user interaction with a z/OS system. The ISPF environment is executed from TSO in native mode. ISPF provides utilities, an editor, and ISPF applications to the

© Copyright IBM Corp. 2001, 2020

user. When permitted by various security controls, you can use ISPF to access most z/OS system functions. TSO/ISPF serves as both a system management interface and a development interface for traditional z/OS programming.

Learning resources:

What is ISPF? (z/OS basic skills)

z/OS ISPF

Db2 Interactive (DB2I)

You can use the included Db2 Interactive (DB2I) set of ISPF panels to work interactively with many different aspects of Db2. For example, it includes panels for interactively issuing SQL statements and commands, running utilities, preparing application programs, setting default options, and more. The following figure shows the options of the DB2I primary options menu.

```
DSNEPRI                     DB2I PRIMARY OPTION MENU          SSID: DSN
COMMAND ===>

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                  (Process SQL statements)
 2  DCLGEN                 (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION    (Prepare a DB2 application program to run)
 4  PRECOMPILE             (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE       (BIND, REBIND, or FREE plans or packages)
 6  RUN                    (RUN an SQL program)
 7  DB2 COMMANDS           (Issue DB2 commands)
 8  UTILITIES              (Invoke DB2 utilities)
 D  DB2I DEFAULTS          (Set global parameters)
 X  EXIT                   (Leave  DB2I)
```

*Figure 42. The ISPF panel for the **DB2I Primary Option Menu***

Learning resources:

“The DB2I primary option menu” on page 173
Issuing commands from TSO terminals (Db2 Administration Guide)
DSN command processor (Db2 Application programming and SQL)

SQL Processor Using File Input (SPUFI)

You can use the *SPUFI* panels of DB2I to issue SQL statements interactively from a TSO/E session.

Learning resources:

Executing SQL by using SPUFI (Db2 Application programming and SQL)

How can I submit and monitor batch work in Db2 for z/OS?

You can use the following facilities to process batch jobs, to start system tasks, and to check the output of jobs and tasks that you submitted to the system:

Job control language (JCL)

You can use *JCL* to tell the system what program to run, followed by a description of program inputs and outputs. Basic JCL contains three types of statements: JOB, EXEC, and DD. A job can contain several EXEC statements (steps) and each step might have several DD statements. JCL provides a wide range of parameters and controls, but you might use only a subset most of the time.

Learning resources:

What is JCL? (z/OS basic skills)

Reusable JCL collection (z/OS basic skills)

Spool Display and Search Facility (SDSF)

You can use the *SDSF* utility to monitor, control, and view the output of jobs in the system. After you submit a job, you can use SDSF to review the output for successful completion or to review and correct JCL errors.

Learning resources:

What is SDSF? (z/OS basic skills)

Related concepts

z/OS basic skills education

Related reference

ABCs of z/OS System Programming: Volume 1 (IBM Redbooks)

Introduction to the New Mainframe: z/OS Basics (IBM Redbooks)

Related information

Db2 basics tutorial: Working with Db2 for z/OS interactively

This tutorial shows you how to use Db2 for z/OS in the z/OS terminal interface to complete some basic database administration tasks.

The DB2I primary option menu

The **DB2I Primary Option Menu** ISPF panel is the starting point for all DB2I tasks.

The following figure shows an example of the ISPF panel for the **DB2I Primary Option Menu**. From this panel, you can access all of the DB2I panels without passing through panels that you do not need.

```
DSNEPRI                    DB2I PRIMARY OPTION MENU          SSID: DSN
COMMAND ===>

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                 (Process SQL statements)
 2  DCLGEN                (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION   (Prepare a DB2 application program to run)
 4  PRECOMPILE            (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE      (BIND, REBIND, or FREE plans or packages)
 6  RUN                   (RUN an SQL program)
 7  DB2 COMMANDS          (Issue DB2 commands)
 8  UTILITIES             (Invoke DB2 utilities)
 D  DB2I DEFAULTS         (Set global parameters)
 X  EXIT                  (Leave  DB2I)
```

*Figure 43. The ISPF panel for the **DB2I Primary Option Menu***

The following descriptions explain the functions on the **DB2I Primary Option Menu** panel.

1 SPUFI

Lets you develop and execute one or more SQL statements interactively. For further information, see Executing SQL by using SPUFI (Db2 Application programming and SQL).

2 DCLGEN

Lets you generate C, COBOL, or PL/I data declarations of tables. For further information, see DCLGEN (declarations generator) (Db2 Application programming and SQL).

3 PROGRAM PREPARATION

Lets you prepare an application program, or prepare and run the application program. For more information, see Db2 Program Preparation panel (Db2 Application programming and SQL).

4 PRECOMPILE

Lets you convert embedded SQL statements into statements that your host language can process. For further information, see Precompile panel (Db2 Application programming and SQL).

5 BIND/REBIND/FREE
Lets you bind, rebind, or free a package or application plan. For more information, see Bind/Rebind/Free Selection panel (Db2 Application programming and SQL).

6 RUN
Lets you run an application program in a TSO or batch environment. For more information, see DB2I Run panel (Db2 Application programming and SQL).

7 DB2 COMMANDS
Lets you issue Db2 commands.

8 UTILITIES
Lets you call Db2 utility programs.

D DB2I DEFAULTS
Lets you set DB2I defaults. See DB2I Defaults Panel 1 (Db2 Application programming and SQL).

X EXIT
Lets you exit DB2I.

Related concepts

TSO attachment facility
You can bind application plans and packages and run several online functions of Db2 through the TSO attachment facility. TSO also enables authorized Db2 users or jobs to create, modify, and maintain databases and application programs.

DSN command processor (Db2 Application programming and SQL)

What is ISPF? (z/OS basic skills)

Related tasks

Submitting work by using DB2I (Db2 Administration Guide)

Issuing commands from TSO terminals (Db2 Administration Guide)

Related reference

DB2I panels that are used for program preparation (Db2 Application programming and SQL)

z/OS ISPF

Related information

Db2 basics tutorial: Working with Db2 for z/OS interactively
This tutorial shows you how to use Db2 for z/OS in the z/OS terminal interface to complete some basic database administration tasks.

Commands for controlling Db2 and related facilities

You can use commands to perform many of the tasks that are required to control and maintain your Db2 subsystem.

For example, you use commands to take the following actions to control Db2 subsystem, and more:

- Start and stop the subsystem and its facilities through commands such as the START DB2 and STOP DB2 commands.
- Request information about the status of the subsystem and its facilities through various DISPLAY commands.
- Modify the configuration of the subsystem through commands such as SET SYSPARM and various ALTER commands
- Control connections to the subsystem from various attachment facilities

For a complete list of commands that you can issue to control the operation of a Db2 subsystem, see About Db2 and related commands (Db2 Commands).

Types of commands for controlling Db2

You can control most aspects of the operation of a Db2 subsystem by using the DSN command of TSO and its subcommands and Db2 commands. However, certain aspects of the Db2 subsystem, such as

connections for various attachement facilities, and the z/OS internal resource lock manager (IRLM) require the use of other types of commands.

You might need to use any of the following *types of commands* to control Db2 operations:

- The TSO command DSN and its subcommands
- Db2 commands
- CICS attachment facility commands
- IMS commands
- Administrative task scheduler commands
- z/OS IRLM commands
- TSO CLISTs

For descriptions of the different command types, see Command types and environments in Db2 (Db2 Commands).

Where and how to enter commands

Within the z/OS environment, you can issue most types of commands from different interactive contexts, including the following consoles and terminals:

- z/OS consoles
- TSO terminals, by any of the following methods:
  - Issuing the DSN command from the TSO READY prompt
  - Entering commands in the **DB2 Commands** panel inDB2I
- IMS terminals
- Authorized CICS terminals

You can issue many commands from the background within batch programs, such as the following types of programs:

- z/OS application programs
- Authorized CICS programs
- IMS programs
- APF-authorized programs, such as a terminal monitor program (TMP)
- IFI application programs

For information of about where and how you can issue commands, see Controlling Db2 operations by using commands (Db2 Administration Guide).

Related concepts

Commands in Db2 (Db2 Commands)

Db2 attachment facilities

An *attachment facility* provides the interface between Db2 and another environment. You can also begin Db2 sessions from other environments on clients such as Microsoft Windows or UNIX by using interfaces like ODBC, JDBC, and SQLJ.

Related tasks

Controlling connections (Db2 Administration Guide)

Submitting work to Db2 (Db2 Administration Guide)

Related reference

Executing the terminal monitor program (TSO/E Customization)

Writing JCL for command execution (TSO/E Customization)

Db2 utilities

IBM Db2 *utilities* help you maintain data in your Db2 for z/OS databases. Generally, you use Db2 utilities to perform large-scale operations across one or more table spaces.

For example, you can use utilities to do the following maintenance tasks:

- Make copies of your data, which can later be used as a basis for recovery.
- Recover data, either to the current time or some point in the past.
- Load thousands or even many millions of rows into your database.
- Reorganize data so that space is used in the most efficient way possible.

Various independent software vendors (ISVs) also provide utilities and tools that you can use to maintain your Db2 databases. However, this topic describes only those Db2 utilities that are provided by IBM.

Where do I get Db2 utilities?

Some utilities are provided with Db2 for z/OS. Other utilities are sold as a separate product called Db2 Utilities Suite for z/OS. For a list of utilities for each of these categories, see Db2 utilities packaging (Db2 Utilities). Your shop might also use utilities that are provided by an ISV. Ask your system programmer which utilities are installed in your shop.

Types of utilities

There are two types of Db2 utilities:

- Online utilities, which require Db2 to be up and running. For more information about online utilities, see Db2 online utilities (Db2 Utilities) and Invoking Db2 online utilities (Db2 Utilities).
- Stand-alone utilities, which run independently of Db2. For more information about stand-alone utilities, see Db2 stand-alone utilities (Db2 Utilities) and Invoking stand-alone utilities (Db2 Utilities).

Commonly used utilities

The following tables list some of the commonly used utilities and a brief description of what they do. Use these tables to help you determine the utilities that you might use in your job.

The following table lists the utilities that you can use to load and unload data and to check data structures.

Table 22. Utilities for checking, loading, and unloading data

Utility	What the utility does
CHECK DATA	Checks table spaces for violations of referential and table check constraints. Also, checks for consistency between a base table space and the corresponding LOB or XML table spaces.
CHECK INDEX	Checks whether indexes are consistent with the data that they index.
CHECK LOB	Identifies any structural defects in a LOB table space and any invalid LOB values.
EXEC SQL	Declares cursors and executes dynamic SQL statements. This utility is typically used as part of the Db2 cross-loader function. The *cross-loader function* transfers data from one location to another location or from one table to another table at the same location with a single LOAD job.
LOAD	Loads data into one or more tables of a table space.
UNLOAD	Copies data from table spaces or image copies in a format that can be used by the LOAD utility. The data is not deleted from the source object.

The following table lists the utilities that are used to maintain the data for optimal query performance and storage.

Table 23. Utilities for maintaining data

Utility	What the utility does
REORG INDEX	Reorganizes an index space to improve access performance and reclaim fragmented space.
REORG TABLESPACE	Reorganizes a table space or partitions to improve access performance and to reclaim fragmented space. REORG is also used to materialize pending metadata changes.
RUNSTATS	Gathers statistics for table spaces, indexes, and partitions and records these statistics in the Db2 catalog. Db2 uses these statistics to select the most efficient access paths for queries.

The following table lists utilities that are used for recovery and planning for recovery.

Table 24. Utilities for recovery

Utility	What the utility does
COPY	Copies table spaces, index spaces, partitions, and data sets to be used for recovery. These copies are called *image copies*. Certain other utilities can also take copies while they run. For example, LOAD can take a copy while it loads data. These copies are called *inline image copies*.
COPYTOCOPY	Copies an image copy or inline image copy.
MERGECOPY	Merges image copies or inline image copies.
QUIESCE	Establishes a quiesce point for table spaces, partitions, and table space sets. A *quiesce point* is a point at which data is consistent across the specified objects and can later be used as a recovery point. (A *table space set* is a set of related objects that should logically be recovered together, for example a base table space and its LOB table spaces.)
RECOVER	Recovers data to the current state or to a previous point in time by restoring a copy and then applying log records.
REPORT	Provides information about table spaces, tables, and indexes that is useful for recovery. For example, REPORT provides the names of related table spaces that should logically be recovered to the same point in time.

The preceding tables list only a subset of the utilities that IBM provides. IBM also provides more advanced utilities, such as those that are used to backup or restore the entire system or repair damaged data or structures. For a complete list of utilities, see Db2 online utilities (Db2 Utilities) and Db2 stand-alone utilities (Db2 Utilities).

How do I run a utility?

The process for running a utility depends on whether you are running an online utility or a stand-alone utility.

Online utilities

Several methods are available for running an online utility. Most likely, you will create a job control language (JCL) job to run a utility. You can also use the DSNUTILU stored procedure (Db2 SQL) stored procedure to run a utility from your application program. Another alternative is to use the Db2 Utilities panel (in DB2I) or the DSNU CLIST command to generate the JCL.

Prior to running an online utility, you should be proficient with the following concepts:

JCL

If you will use JCL to run utilities and need to learn more about it, use the following resources:

- For a brief overview of JCL, see What is JCL? (z/OS basic skills).
- For a more extensive introduction to JCL that includes exercises for coding JCL, submitting a job, and viewing the result, see Introduction to the New Mainframe: z/OS Basics (IBM Redbooks).
- To practice using JCL, complete the exercises in Reusable JCL collection (z/OS basic skills).
- For in-depth JCL training, take the course "z/OS Job Control Language and Utilities."" Go to IBM Training Search and search for this course.

Utility control statements

After you master JCL, you should understand how to write a utility control statement. A *utility control statement* specifies the utility name, the target object or objects, and any other options for the utility. For example, the following utility control statement is for the CHECK DATA utility:

```
CHECK DATA TABLESPACE DBIQUQ01.TPIQUQ01 SCOPE ALL
      AUXERROR INVALIDATE
```

The options that are available for each utility are listed in the syntax diagram in the documentation for that utility. For example, the syntax diagram for the CHECK DATA utility is in Syntax and options of the CHECK DATA control statement (Db2 Utilities). If you are not familiar with syntax diagrams, read How to read syntax diagrams.

For detailed rules for writing utility control statements, such as how you continue statements across lines and which encoding schemes are acceptable, read Utility control statements (Db2 Utilities).

Where you create the utility statement depends on the procedure that you use to invoke the utility. For example, if you JCL, the utility statement is specified in the SYSIN DD statement. If you use the DSNUTILU procedure, the utility statement gets passed as a parameter. If you use the Db2 Utilities panel or the DSNU CLIST command, you can put this utility statement in a data set or member. See Invoking Db2 online utilities (Db2 Utilities) for detailed instructions.

Data sets

Because many utilities require input data sets or work data sets or allocate output data sets, you should understand how to allocate and use data sets.

Which data sets are required depends on the utility and the options that you specify. For example, suppose you request that the COPY utility take a primary and backup copy at the local site and a primary and backup copy at the recovery site. In this case, COPY allocates four data sets, one for each image copy, and you must tell COPY the parameters to use for these output data sets. For details about which data sets are required for each utility, see the documentation for the utility. For example, the data sets that are required by COPY are described in Data sets that COPY uses (Db2 Utilities) uses.

DD statements, DD names, and DD statement parameters

- For information about DD statements and DD names (also sometimes called ddnames), see JCL statements: What does the DD statement do? (z/OS basic skills) and JCL DD statement: ddnames that are reserved for specific uses (z/OS basic skills).
- For information about basic DD statement parameters, see JCL DD statements: Positional and frequently used parameters (z/OS basic skills).

Templates

Db2 online utilities provide a TEMPLATE control statement, which you can use instead of JCL DD statements to define data sets. After a template is defined, its name is passed to a utility by using the same keywords as a DD name. For example, in the following statement, MYCOPY can refer to either a JCL DD name or a template name:

```
COPY TABLESPACE A.B COPYDDN MYCOPY
```

See “LISTDEF, OPTIONS, and TEMPLATE utilities” on page 179 for more information about templates.

For complete instructions for running online utilities, see Invoking Db2 online utilities (Db2 Utilities).

Stand-alone utilities

To run a stand-alone utility, you must use JCL. You specify the utility name on the EXEC statement. For certain utilities, you also specify the utility options on the EXEC statement. For other utilities, you specify the options in a utility control statement in an in-stream data set.

Important: Most stand-alone utilities are intended for advanced Db2 users. Be sure that you understand the effects of running a stand-alone utility before you run one.

For the complete instructions for running a stand-alone utility, see Invoking stand-alone utilities (Db2 Utilities).

Required authorizations

For both online and stand-alone utilities, you must have the correct authorization to run them. The documentation for each utility describes the authorization that is required for that utility. They are not all the same. For more information about general authorization for utilities, see Privileges and authorization IDs for Db2 utilities (Db2 Utilities).

LISTDEF, OPTIONS, and TEMPLATE utilities

The LISTDEF, OPTIONS, and TEMPLATE utilities are different from the other utilities in that they do not perform any database administration function. Instead, they provide facilities to the other utilities.

- The LISTDEF utility provides a facility for producing lists of objects on which to operate.
- The OPTIONS utility provides a facility for globally setting options that apply to the entire job step.
- The TEMPLATE utility provides a facility for allocating data sets.

LISTDEF

You can use the LISTDEF utility to create a reusable list of objects. For example, suppose that you have a group of objects that you want to regularly copy, quiesce, or perform any other utility task on. You can use the LISTDEF utility to create a list of those objects. You specify the object names and then give the list a name. For example, the following statement defines a list called MYLIST, which includes three objects:

```
LISTDEF MYLIST INCLUDE TABLESPACE A
               INCLUDE TABLESPACE B
               INCLUDE TABLESPACES TABLE C
```

Your lists will likely include many more objects. If the objects have a similar naming pattern, you can use wildcard characters for portions of the object names.

After you define the list, you can then reference that list in subsequent utility control statements. For example, the following COPY statement requests that all of the objects in your list (MYLIST) be copied:

```
COPY LIST MYLIST …
```

Using lists is helpful when you need to run utilities on many objects. Instead of looking up all of the object names in the catalog and writing each object name in a control statement, you can define a list. You can then reference that list in any subsequent utility control statements.

You can either include the LISTDEF statement in the SYSIN DD statement directly before the utility control statements that reference it, or you can create libraries of LISTDEF statements. *LISTDEF libraries* are just data sets that contain LISTDEF statements. The advantage of creating libraries is that these lists are then stored and available for many different utility jobs.

To learn more about the LISTDEF utility, see LISTDEF (Db2 Utilities).

OPTIONS

You can use the OPTIONS utility to specify utility processing options for the job step. Consider the following examples:

- You can use the OPTIONS utility to specify a LISTDEF library or TEMPLATE library that any subsequent utilities are to use.
- You can specify OPTIONS PREVIEW, which causes the utility to run in preview mode. Preview mode is useful to view what the utility job will do before it is actually done. For example, if you specify OPTIONS PREVIEW and then a LISTDEF statement, you can see the expanded list of all objects that are included in the list. Such an expanded list is useful if your list contains wildcard characters, and you want to see which objects will be included in the list.
- You can also use the OPTIONS utility to specify what the utility will do if an error occurs for an item in the list. You can specify whether the utility is to stop processing or skip the item in error and continue processing the next item.

To learn more about the OPTIONS utility, see OPTIONS (Db2 Utilities).

TEMPLATE
You can use the TEMPLATE utility to allocate data sets without using JCL DD statements. The TEMPLATE utility defines data set templates. For each template, you specify the template name, the data set naming convention, and any allocation parameters that you want. Then, you can tell subsequent utilities to use that template when they allocate certain data sets.

One advantage of using this utility is that you do not have to specify data set allocation parameters. For example, TEMPLATE can determine the appropriate data set size for you.

Templates are also useful for lists of objects. For example, suppose that you have a long list of objects to copy. Instead of using a DD statement to specify a data set for each image copy, you can specify the template name where you otherwise would specify a DD name.

To learn more about the TEMPLATE utility, see TEMPLATE (Db2 Utilities).

How do I monitor a utility job?

You can monitor the progress of one or more utility jobs by using the DISPLAY UTILITY command. For example, you can see what phase the utility is in and the object that it last processed.

If you need to stop an online utility job, use the TERM UTILITY command.

If a utility job stopped before it completed processing, you can restart it after you fix the error that caused the utility job to stop. You can restart a utility either from the last checkpoint or from the beginning of the phase in which it was currently processing.

For more information about monitoring, stopping, and restarting utilities, see Monitoring and controlling online utilities (Db2 Utilities).

Tools

You can use tools that are provided by IBM or various ISVs to perform the tasks that are required to control and maintain your Db2 subsystem. For example, you can use the following IBM Db2 Tools for z/OS tools to manage utilities and perform utility tasks:

Db2 Automation Tool for z/OS
A tool that enables database administrators to focus on database optimization. This tool automates maintenance tasks. See Db2 Automation Tool for z/OS .

Db2 High Performance Unload for z/OS
A high-speed Db2 utility that unloads Db2 tables from either a table space or a backup of the database. See Db2 High Performance Unload for z/OS .

Db2 Cloning Tool for z/OS
A tool that quickly clones a Db2 subsystem or data sets, creating the equivalent of a production environment that you can use to test new features and functions. See Db2 Cloning Tool for z/OS .

Db2 Sort for z/OS
A tool that reduces elapsed time and CPU cost for utility sort processing. See Db2 Sort for z/OS .

Other learning resources

For more detailed training in Db2 utilities, use the following resources:

- For an IBM Redpaper publication that helps Db2 database administrators, Db2 system programmers, and anyone who runs Db2 for z/OS utilities implement best practices, see Db2 for z/OS Utilities in Practice.
- . For a tutorial module that shows you two methods for running Db2utilities, including using a Db2 Interactive (DB2I) panel and writing job control language (JCL) jobs, see “Module 2: Running Db2 utilities” on page 353
- Although this IBM Redbooks publication was written for DB2 9, the basics are the same for later Db2 versions. See, DB2 9 for z/OS: Using the Utilities Suite (IBM Redbooks)
- Course: "z/OS and Db2 Basics for Db2 for z/OS DBA Beginners." You can find links to this course information from the Db2 for z/OS training paths. Both classroom and online options are available.
- Db2 Utilities (Db2 for z/OS Best Practices)

Management of data sets

In Db2 for z/OS, one way that you manage data sets is by using Storage Management Subsystem (SMS).

Table spaces or indexes that exceed 4 GB in size require SMS-managed data sets. Other table spaces and indexes can be stored in user-managed data sets or in Db2-managed storage groups.

Related concepts

Db2 and DFSMS
You can use the DFSMSdfp Storage Management Subsystem (SMS) to manage Db2 disk data sets.

Assignment of table spaces to physical storage
You can store table spaces and index spaces in user-managed storage, SMS-managed storage, or in Db2-managed storage groups. (A *storage group* is a set of disk volumes.)

Authorization and security mechanisms for data access

Authorization is an important part of controlling Db2. The security and authorization mechanisms that control access to Db2 data are both direct and indirect.

Db2 performs direct security checks of user IDs and passwords before users gain access through DDF. All other attachment facilities require that the user authenticate with DDF before attaching to Db2. Db2 security mechanisms include specific objects, privileges on those objects, and some privileges that provide broader authority. Db2 also controls data access indirectly with authorization checks at bind time and run time for application plans and packages.

Authorization

You probably noticed references to authorization in this information. For example, you must be authorized to run SQL statements that create and alter Db2 objects. Even when users run a SELECT statement to query table information, their authorization might limit what they see. The user might see data only in a subset of columns that are defined in a view. Views provide a good variety of security controls.

Before you issue Db2 commands, run utilities, run application packages and plans, or use most other Db2 functions, you need the appropriate authorization or privilege. For example, to make changes to a table, you need authorization to access that table. A *privilege* allows an action on an object. For example, to insert data into a table requires the privilege to insert data.

GRANT and REVOKE statements provide access control for Db2 objects. Privileges and authorities can be granted to authorization IDs and roles in many combinations, and they can also be revoked.

You can use the RACF component or an equivalent product to control access to Db2 objects. This is the best option if you want the z/OS security administrator to manage access to data instead for the database administrator.

Security

Due to the need for greater data security and demands for improved corporate accountability, the federal government and certain industries have developed laws and regulations to guide many corporate processes. The expectation to comply with these laws and regulations is likely to increase in the future. Db2 for z/OS support of roles and trusted contexts help in the area of compliance by enforcing data accountability at the data level. Instead of using a single user ID for all database requests, application servers can provide an user ID with no performance penalty associated with the request.

Related concepts

Db2 and the z/OS Security Server
The z/OS Security Server prevents unauthorized system access and can protect Db2 resources, such as tables. The z/OS Security Server is sometimes referred to as RACF, which is one of its key components.

How authorization IDs control data access

One of the ways that Db2 controls access to data is through the use of identifiers. A set of one or more Db2 identifiers, called *authorization IDs*, represents every process that connects to or signs on to Db2.

Authorization IDs come in three types:

Primary authorization ID
As a result of assigning authorization IDs, every process has exactly one ID, called the *primary authorization ID*. Generally, the primary authorization ID identifies a process. For example, statistics and performance trace records use a primary authorization ID to identify a process.

Secondary authorization ID
All other IDs are *secondary authorization IDs*. A secondary authorization ID, which is optional, can hold additional privileges that are available to the process. For example, you could use a secondary authorization ID for a z/OS Security Server group.

CURRENT SQLID
One ID (either primary or secondary) is designated as the *CURRENT SQLID*. The CURRENT SQLID holds the privileges that are exercised when certain dynamic SQL statements run. You can set the CURRENT SQLID to the primary ID or to any of the secondary IDs. If an authorization ID of a process has system administration (SYSADM) authority, the process can set its CURRENT SQLID to any authorization ID. You can change the value of the CURRENT SQLID during your session.

Example: If ALPHA is your primary authorization ID or one of your secondary authorization IDs, you can make it the CURRENT SQLID by issuing this SQL statement:

```
SET CURRENT SQLID = 'ALPHA';
```

Related concepts

How authorization IDs hold privileges and authorities
Db2 controls access to its objects by using a set of privileges. Each privilege allows an action on some object.

Ways to control access to Db2 objects through explicit privileges and authorities
You can control access to Db2 user data by granting, not granting, or revoking explicit privileges and authorities.

Ways to control access to data
Db2 enables you to control data access. Access to data includes a user who is engaged in an interactive terminal session. For example, access can be from a remote server, from an IMS or a CICS transaction, or from a program that runs in batch mode.

How authorization IDs hold privileges and authorities

Db2 controls access to its objects by using a set of privileges. Each privilege allows an action on some object.

The following figure shows the primary ways within Db2 to give access to data to an ID.

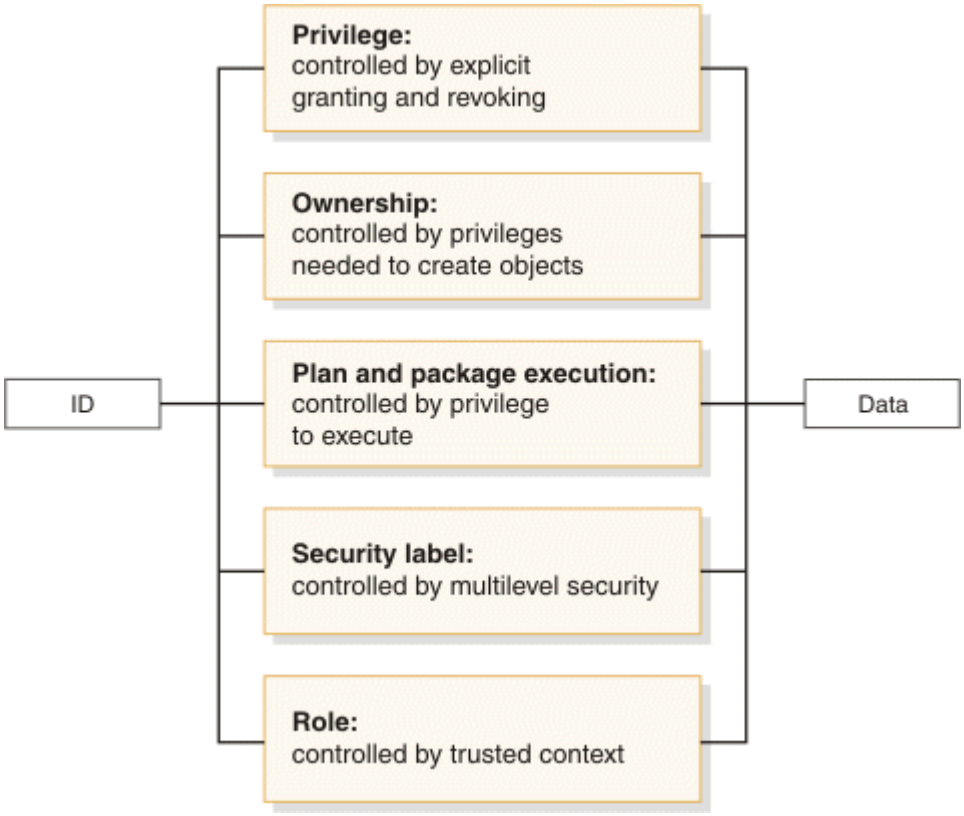


Figure 44. Granting access to data within Db2

IDs can hold privileges that allow them to take certain actions or be prohibited from doing so. Db2 privileges provide extremely precise control.

Related privileges
Db2 defines sets of related privileges, called *administrative authorities*. By granting an administrative authority to an ID, you grant all the privileges that are associated with it, in one statement.

Object privileges
Ownership of an object carries with it a set of related privileges on the object. An ID can own an object that it creates, or it can create an object that another ID is to own. Creation and ownership of objects are separately controlled.

Application plan and package privileges
The privilege to execute an application plan or a package deserves special attention. Executing a plan or package implicitly exercises all the privileges that the plan or package owner needed when binding it. Therefore, granting the privilege to execute can provide a detailed set of privileges and can eliminate the need to grant other privileges separately.

Example: Assume that an application plan issues the INSERT and SELECT statement on several tables. You need to grant INSERT and SELECT privileges only to the plan owner. Any authorization ID that is subsequently granted the EXECUTE privilege on the plan can perform those same INSERT and SELECT statements through the plan. You don't need to explicitly grant the privilege to perform those statements to that ID.

Security labels
Multilevel security restricts access to an object or a row based on the security label of the object or row and the security label of the user.

Roles
A *role* is a database entity that groups together one or more privileges. A role is available only when the process is running in a trusted context. A *trusted context* is a database entity that is based on a system authorization ID and a set of connection trust attributes. You can create and use a trusted

context to establish a trusted connection between Db2 and an external entity, such as a middleware server.

Users are associated with a role in the definition of a trusted context. A trusted context can have a default role, specific roles for individual users, or no roles at all.

Related concepts

How authorization IDs hold privileges and authorities
Db2 controls access to its objects by using a set of privileges. Each privilege allows an action on some object.

Ways to control access to Db2 objects through explicit privileges and authorities
You can control access to Db2 user data by granting, not granting, or revoking explicit privileges and authorities.

Ways to control access to data
Db2 enables you to control data access. Access to data includes a user who is engaged in an interactive terminal session. For example, access can be from a remote server, from an IMS or a CICS transaction, or from a program that runs in batch mode.

Ways to control access to Db2 subsystems

Db2 for z/OS performs security checks to authenticate users before they gain access to Db2 data. A variety of authentication mechanisms are supported by Db2 requesters and accepted by Db2 servers.

Authentication occurs when the CONNECT statement is issued to connect the application process to the designated server. The server or the local Db2 subsystem checks the authorization ID and password to verify that the user is authorized to connect to the server.

You can use RACF or the z/OS Security Server to authenticate users that access a Db2 database.

Related concepts

Db2 and the z/OS Security Server
The z/OS Security Server prevents unauthorized system access and can protect Db2 resources, such as tables. The z/OS Security Server is sometimes referred to as RACF, which is one of its key components.

Local Db2 access

A local Db2 user is subject to several security checks.

For example, when Db2 runs under TSO and use the TSO logon ID as the Db2 primary authorization ID, that ID is verified with a password when the user logs on.

When the server is the local Db2 subsystem, RACF verifies the password and checks whether the authorization ID is allowed to use the Db2 resources that are defined to RACF. If an exit routine is defined, RACF or the z/OS Security Server perform additional security checking.

Remote Db2 access

When the server is not the local Db2 subsystem, multiple security checks occur.

- The local security manager at the server verifies the Db2 primary authorization ID and password. A subsequent verification determines whether the authorization ID is allowed to access Db2.
- Security options for SNA or TCP/IP protocols are checked in the communications database (CDB).

DDF supports TCP/IP and SNA communication protocols in a distributed environment. As a requester or a server, Db2 chooses how to send or accept authentication mechanisms, based on which network protocol is used. Db2 uses SNA security mechanisms for SNA network connections and DRDA security mechanisms for TCP/IP or Kerberos network connections.

DRDA security options provide the following support for encrypting sensitive data:

- Db2 for z/OS servers can provide secure, high-speed data encryption and decryption.
- Db2 for z/OS requesters have the option of encrypting user IDs and passwords when requesters connect to remote servers. Requesters can also encrypt security-sensitive data when communicating with servers so that the data is secure when traveling over the network.

You can use RACF or a similar security subsystem to perform authentication. RACF can:

- Verify a remote authorization ID associated with a connection by checking the ID against its password.
- Verify whether the authorization ID is allowed to access Db2 through a remote connection.
- Verify whether the authorization ID is allowed to access Db2 from a specific remote site.
- Generate PassTickets, an alternative to passwords, on the sending side. A *PassTicket* lets a user gain access to a host system without sending the RACF password across the network.

Kerberos security

As a server, Db2 supports Kerberos security for authenticating remote users. The authentication mechanisms are encrypted Kerberos tickets rather than user IDs and passwords.

You can establish Db2 for z/OS support for Kerberos authentication through the z/OS Security Server. Kerberos is also a network security option for Db2 Connect clients.

Communications database

The Db2 communications database contains a set of Db2 catalog tables that let you control aspects of remote requests. Db2 uses this database to obtain information about connections with remote systems.

Workstation access

When a workstation client accesses a Db2 for z/OS server, Db2 Connect passes all authentication information from the client to the server. Workstation clients can encrypt user IDs and passwords when they issue a CONNECT statement. Database connection services (DCS) authentication must be set to DCS_ENCRYPT.

An authentication type for each instance determines user verification. The authentication type is stored in the database manager configuration file at the server. The following authentication types are allowed with Db2 Connect:

CLIENT
The user ID and password are validated at the client.

SERVER
The user ID and password are validated at the database server.

SERVER_ENCRYPT
The user ID and password are validated at the database server, and passwords are encrypted at the client.

KERBEROS
The client logs onto the server by using Kerberos authentication.

Ways to control access to data

Db2 enables you to control data access. Access to data includes a user who is engaged in an interactive terminal session. For example, access can be from a remote server, from an IMS or a CICS transaction, or from a program that runs in batch mode.

This information discusses different methods of data access control in Db2. In this information, the term *process* is used to represent all forms of access to data.

The following figure suggests several routes from a process to Db2 data, with controls on every route.

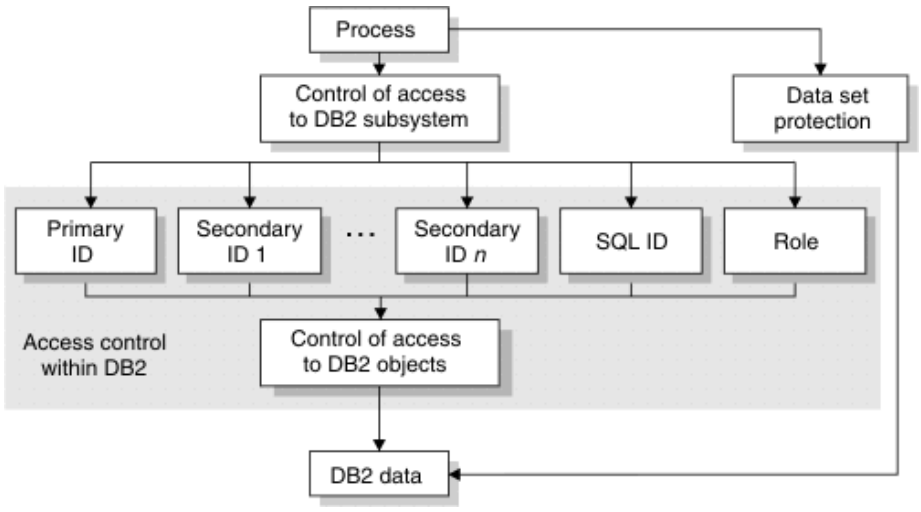


Figure 45. Db2 data access control

The first method, access control within Db2, uses identifiers (IDs) to control access to Db2 objects. The process must first satisfy the security requirements to access the Db2 subsystem. When the process is within the Db2 subsystem, Db2 checks various IDs to determine whether the process can access Db2 objects. These IDs (primary authorization ID, secondary authorization ID, and SQL ID) are described. If the process has the necessary ID or IDs, it can access Db2 objects, including Db2 data.

The second method, data set protection, is not controlled within Db2. The process goes through data set protection outside of Db2. If the process satisfies the protection criteria, it reaches the Db2 data.

Related concepts

How authorization IDs control data access
One of the ways that Db2 controls access to data is through the use of identifiers. A set of one or more Db2 identifiers, called *authorization IDs*, represents every process that connects to or signs on to Db2.

How authorization IDs hold privileges and authorities
Db2 controls access to its objects by using a set of privileges. Each privilege allows an action on some object.

Ways to control access to Db2 objects through explicit privileges and authorities
You can control access to Db2 user data by granting, not granting, or revoking explicit privileges and authorities.

Managing access through authorization IDs and roles (Managing Security)

Ways to control access to Db2 objects through explicit privileges and authorities

You can control access to Db2 user data by granting, not granting, or revoking explicit privileges and authorities.

An *explicit privilege* is a named privilege that is granted with the GRANT statement or that is revoked with the REVOKE statement. An administrative authority is a set of privileges, often encompassing a related set of objects. Authorities often include privileges that are not explicit, have no name, and cannot be individually granted, such as the ability to terminate any utility job.

Explicit privileges

Explicit privileges provide detailed control. For example, assume that a user needs to select, insert, and update data in a table. To complete these actions, the user needs the SELECT, INSERT, and UPDATE privilege on the table.

Explicit privileges are available for these objects:

- Buffer pools
- Collections

- Databases
- Distinct types
- JARs (a Java Archive, which is a file format for aggregating many files into one file)
- Packages
- Plans
- Routines (functions and procedures)
- Schemas
- Sequences
- Storage groups
- Systems
- Tables
- Table spaces
- Views

Administrative authorities

Privileges are grouped into administrative authorities. Those authorities form a hierarchy. Each authority includes a specific group of privileges. The administrative authorities fall into the categories of system, database, and collection authorities. The highest-ranking administrative authority is SYSADM. Each level of authority includes the privileges of all lower-ranking authorities.

The following system authorities are ranked from highest to lowest:

SYSADM

System administration authority includes all Db2 privileges (except for a few that are reserved for installation), which are all grantable to others.

You can limit the ability of SYSADM to manage access to roles. You can also limit the ability of SYSADM to grant and revoke authorities and privileges.

SYSCTRL
System control authority includes most SYSADM privileges, but it excludes the privileges to read or change user data.

SYSOPR
System operator authority includes the privileges to issue most Db2 commands and to terminate any utility job.

The following database authorities are ranked from highest to lowest:

DBADM
Database administration authority includes the privileges to control a specific database. Users with DBADM authority can access tables and alter or drop table spaces, tables, or indexes in that database.

DBCTRL
Database control authority includes the privileges to control a specific database and run utilities that can change data in the database.

DBMAINT
Database maintenance authority includes the privileges to work with certain objects and to issue certain utilities and commands in a specific database.

Additional administrative authorities include the following:

ACCESSCTRL
Access control authority allows SECADM to delegate the ability to grant and revoke object privileges and most administrative authorities.

DATAACCESS
Data access authority controls DBA access to user data in Db2.

EXPLAIN
EXPLAIN authority allows a user to issue EXPLAIN, PREPARE, and DESCRIBE statements without requiring the privilege to execute the statement.

PACKADM
Package administrator authority gives access to designated collections.

SECADM
Security administrator authority allows a user to manage access to a table in Db2, but cannot create, alter or drop a table.

SQLADM
SQL administrator authority provides the ability to monitor and tune SQL without any additional privileges.

Related concepts

How authorization IDs control data access
One of the ways that Db2 controls access to data is through the use of identifiers. A set of one or more Db2 identifiers, called *authorization IDs*, represents every process that connects to or signs on to Db2.

How authorization IDs hold privileges and authorities
Db2 controls access to its objects by using a set of privileges. Each privilege allows an action on some object.

Row-level and column-level access control

You can use row-level and column-level access control to restrict access to certain types of information that require additional security.

Row-level and column-level access controls can help you to protect sensitive information and comply with government regulations for security and privacy. These access controls work with explicit privileges and administrative authorities. If you use row-level or column-level access control, view level access control is unnecessary.

Db2 restricts access to columns and rows based upon individual user permissions. When Db2 is in new function mode, the SECADMIN authority manages the privacy and security policies that are associated with individual tables. The SECADMIN authority also grants and revokes access privileges to specific rows and columns. Row-level and column-level access control affects all users and database administrators.

Row-level and column-level access control provides the following advantages:

- Integration within the database system
- Database level security
- SQL enforced security that does not require other products to monitor access
- Access that is managed by the Db2 security administrator
- Multiple access levels based on users, groups, or roles
- Row-level and column-level access control with filtering and data masking
- No requirement to filter sensitive data at the application level

Use of multilevel security to control access

Db2 provides a powerful security scheme called multilevel security. *Multilevel security* is a security policy that classifies data and users according to a system of hierarchical security levels and nonhierarchical security categories.

Multilevel security prevents unauthorized users from accessing information at a higher classification than their authorization, and it prevents users from declassifying information.

Using multilevel security, you can define security for Db2 objects and perform other checks, including row-level security checks. Row-level security checks control which users have authorization to view, modify, or perform actions on table rows. With multilevel security, you do not need to use special views or database variables to control security at the row level.

You can create a security label for a table row by defining a column in the CREATE TABLE or ALTER TABLE statement as the security label. As each row is accessed, Db2 uses RACF to compare the security label of the row and the user to determine if the user has appropriate authorization to access the row. Row-level security checks occur whenever a user issues a SELECT, INSERT, UPDATE, or DELETE statement to access a table with a security-label column or runs a utility request for data in a row that is protected by a security label.

Related reference

Implementing multilevel security with Db2 (Managing Security)

Use of views to control access

The table privileges DELETE, INSERT, SELECT, and UPDATE can also be granted on a view. By creating a view and granting privileges on it, you can give an ID access to only a specific subset of data. This capability is sometimes called *field-level access control* or *field-level sensitivity*.

Example: Suppose that you want a particular ID, say MATH110, to be able to extract certain data from the EMP table for statistical investigation. To be exact, suppose that you want to allow access to data like this:

- From columns HIREDATE, JOB, EDL, SALARY, COMM (but not an employee's name or identification number)
- Only for employees that were hired after December 15, 1996
- Only for employees with an education level of 14 or higher
- Only for employees whose job is **not** MGR or PRS

You can create and name a view that shows exactly that combination of data:

GUPI

```
CREATE VIEW SALARIES AS
   SELECT HIREDATE, JOB, EDL, SALARY, COMM
      FROM EMP
      WHERE HIREDATE> '1996-12-15' AND EDLEVEL>= 14
      AND JOB IS DISTINCT FROM 'MGR' AND JOB IS DISTINCT FROM 'PRS';
```

GUPI

Then you can use the GRANT statement to grant the SELECT privilege on the view SALARIES to MATH110:

```
GRANT SELECT ON SALARIES TO MATH110;
```

Now, MATH110 can run SELECT statements that query only the restricted set of data.

Related concepts

A view that combines information from several tables

You can create a view that contains a union of more than one table. A union of more than one table is called a *join*.

Use of grant and revoke privileges to control access

The SQL GRANT statement lets you grant explicit privileges to authorization IDs. The REVOKE statement lets you take them away. Only a privilege that has been explicitly granted can be revoked.

Granting privileges is very flexible. For example, consider table privileges. You can grant all the privileges on a table to an ID. Alternatively, you can grant separate, specific privileges that allow that ID to retrieve data from the table, insert rows, delete rows, or update specific columns. By granting or not granting those privileges on views of the table, you can effectively determine exactly what action an ID can or cannot take on the table.

You can use the GRANT statement to assign privileges as follows:

- Grant privileges to a single ID or to several IDs in one statement.

- Grant a specific privilege on one object in a single statement, grant a list of privileges, or grant privileges over a list of objects.
- Grant ALL, for all the privileges of accessing a single table or for all privileges that are associated with a specific package.

Examples of grant privileges

The following examples show how to grant some system privileges, use privileges, and table privileges.

GUPI

Grant example 1: To grant the privileges of system operator authority to user NICHOLLS, the system administrator uses the following statement:

```
GRANT SYSOPR TO NICHOLLS;
```

Assume that your business decides to associate job tasks with authorization IDs.

Grant example 2: In the following examples, PKA01 is the ID of a package administrator, and DBA01 is the ID of a database administrator. Suppose that the system administrator uses the ADMIN authorization ID, which has SYSADM authority, to issue the following GRANT statements:

- `GRANT PACKADM ON COLLECTION GOLFS TO PKA01 WITH GRANT OPTION;`

  This statement grants PACKADM authority to PKA01. PKA01 acquires package privileges on all packages in the collection named GOLFS and the CREATE IN privilege on that collection. In addition, specifying WITH GRANT OPTION gives PKA01 the ability to grant those privileges to others.
- `GRANT CREATEDBA TO DBA01;`

  CREATEDBA grants DBA01 the privilege to create databases, and DBA01 acquires DBADM authority over those databases.
- `GRANT USE OF STOGROUP SG1 TO DBA01 WITH GRANT OPTION;`

  This statement allows DBA01 to use storage group SG1 and to grant that privilege to others.
- `GRANT USE OF BUFFERPOOL BP0, BP1 TO DBA01 WITH GRANT OPTION;`

  This statement allows DBA01 to use buffer pools BP0 and BP1 and to grant that privilege to others.

Grant example 3: The following examples show specific table privileges that you can grant to users.

- `GRANT SELECT ON DEPT TO PUBLIC;`

  This statement grants SELECT privileges on the DEPT table. Granting the select privilege to PUBLIC gives the privilege to all users at the current server.
- `GRANT UPDATE (EMPNO,DEPT) ON TABLE EMP TO NATZ;`

  This statement grants UPDATE privileges on columns EMPNO and DEPT in the EMP table to user NATZ.
- `GRANT ALL ON TABLE EMP TO KWAN,ALONZO WITH GRANT OPTION;`

  This statement grants all privileges on the EMP table to users KWAN and ALONZO. The WITH GRANT OPTION clause allows these two users to grant the table privileges to others.

GUPI

Examples of revoke privileges

The same ID that grants a privilege can revoke it by issuing the REVOKE statement. If two or more grantors grant the same privilege to an ID, executing a single REVOKE statement does not remove the privilege for that ID. To remove the privilege, each ID that explicitly granted the privilege must explicitly revoke it.

Here are some examples of revoking privileges that were previously granted.

GUPI

Revoke example 1:

- `REVOKE SYSOPR FROM NICHOLLS;`

  This statement revokes SYSOPR authority from user NICHOLLS.
- `REVOKE UPDATE ON EMP FROM NATZ;`

  This statement revokes the UPDATE privilege on the EMP table from NATZ.
- `REVOKE ALL ON TABLE EMP FROM KWAN,ALONZO;`

  This statement revokes all privileges on the EMP table from users KWAN and ALONZO.

An ID with SYSADM or SYSCTRL authority can revoke privileges that are granted by other IDs.

Revoke example 2: A user with SYSADM or SYSCTRL authority can issue the following statements:

- `REVOKE CREATETAB ON DATABASE DB1 FROM PGMR01 BY ALL;`

  In this statement, the CREATETAB privilege that user PGMR01 holds is revoked regardless of who or how many people explicitly granted this privilege to this user.
- `REVOKE CREATETAB, CREATETS ON DATABASE DB1 FROM PGMR01 BY DBUTIL1;`

  This statement revokes privileges that are granted by DBUTIL1 and leaves intact the same privileges if they were granted by any other ID.

GUPI

Revoking privileges can be more complicated. Privileges can be revoked as the result of a cascade revoke. In this case, revoking a privilege from a user can also cause that privilege to be revoked from other users.

Related reference

GRANT (Db2 SQL)

REVOKE (Db2 SQL)

Backup, recovery, and restart

Although high availability of data is a goal for all Db2 subsystems, unplanned outages are difficult to avoid entirely. A good backup, recovery, and restart strategy, however, can reduce the elapsed time of an unplanned outage.

To reduce the probability and duration of unplanned outages, you should periodically back up and reorganize your data to maximize the availability of data to users and programs.

Many factors affect the availability of the databases. Here are some key points to be aware of:

- You should understand the options of utilities such as COPY and REORG.
  - You can recover online such structures as table spaces, partitions, data sets, a range of pages, a single page, and indexes.
  - You can recover table spaces and indexes at the same time to reduce recovery time.
  - With some options on the COPY utility, you can read and update a table space while copying it.
- I/O errors have the following affects:
  - I/O errors on a range of data do not affect availability to the rest of the data.
  - If an I/O error occurs when Db2 is writing to the log, Db2 continues to operate.
  - If an I/O error is on the active log, Db2 moves to the next data set. If the error is on the archive log, Db2 dynamically allocates another data set.
- Documented disaster recovery methods are crucial in the case of disasters that might cause a complete shutdown of your local Db2 subsystem.

- If Db2 is forced to a single mode of operations for the bootstrap data set or logs, you can usually restore dual operation while Db2 continues to run.

Db2 provides extensive methods for recovering data after errors, failures, or even disasters. You can recover data to its current state or to an earlier state. The units of data that can be recovered are table spaces, indexes, index spaces, partitions, and data sets. You can also use recovery functions to back up an entire Db2 subsystem or data sharing group.

Development of backup and recovery procedures is critical in preventing costly and time-consuming data losses. In general, ensure that the following procedures are in place:

- Create a point of consistency.
- Restore system and data objects to a point of consistency.
- Back up and recover the Db2 catalog and your data.
- Recover from out-of-space conditions.
- Recover from a hardware or power failure.
- Recover from a z/OS component failure.

In addition, your site should have a procedure for recovery at a remote site in case of disaster.

Specific problems that require recovery might be anything from an unexpected user error to the failure of an entire subsystem. A problem can occur with hardware or software; damage can be physical or logical. Here are a few examples:

- If a system failure occurs, a restart of Db2 restores data integrity. For example, a Db2 subsystem or an attached subsystem might fail. In either case, Db2 automatically restarts, backs out uncommitted changes, and completes the processing of committed changes.
- If a media failure (such as physical damage to a data storage device) occurs, you can recover data to the current point.
- If data is logically damaged, the goal is to recover the data to a point in time before the logical damage occurred. For example, if Db2 cannot write a page to disk because of a connectivity problem, the page is logically in error.
- If an application program ends abnormally, you can use utilities, logs, and image copies to recover data to a prior point in time.

Recovery of Db2 objects requires adequate image copies and reliable log data sets. You can use a number of utilities and some system structures for backup and recovery. For example, the REPORT utility can provide some of the information that is needed during recovery. You can also obtain information from the bootstrap data set (BSDS) inventory of log data sets.

Related tasks

Recovering from different Db2 for z/OS problems (Db2 Administration Guide)

Related reference

COPY (Db2 Utilities)

RECOVER (Db2 Utilities)

REORG TABLESPACE (Db2 Utilities)

REPORT (Db2 Utilities)

Related information

Operation and recovery (Db2 Administration Guide)

Backup and recovery resources and tools

Db2 relies on the log and the BSDS to record data changes as they occur. The log and BSDS provide critical information during recovery. Other important tools that you need for backup and recovery of data are several of the Db2 utilities.

Log usage

The Db2 log registers data changes and significant events as they occur. Db2 writes each log record to the active log, which is a disk data set. When the active log data set is full, Db2 copies its contents to the archive log, which is a disk or a tape data set. This process is called *offloading*.

The archive log can consist of up to 10000 data sets. Each archive log is a sequential data set (physical sequential) that resides on a disk or magnetic tape volume.

With Db2, you can choose either single logging or dual logging. A single active log contains up to 93 active log data sets. With dual logging, Db2 keeps two identical copies of the log records. Dual logging is the better choice for increased availability.

Bootstrap data set usage

The *bootstrap data set (BSDS)* is a repository of information about the data sets that contain the log. The BSDS contains the following information:

- An inventory of all active and archive log data sets that are known to Db2.

  Db2 records data about the log data set in the BSDS each time a new archive log data set is defined or an active log data set is reused. The BSDS inventory includes the time and date that the log was created, the data set name, its status, and other information. Db2 uses this information to track the active and archive log data sets. Db2 also uses this information to locate log records for log read requests that occur during normal Db2 subsystem activity and during restart and recovery processing.
- An inventory of all recent checkpoint activity that Db2 uses during restart processing.
- A distributed data facility (DDF) communication record.
- Information about buffer pools.

Because the BSDS is essential to recovery in the event of a subsystem failure, Db2 automatically creates two copies of the BSDS during installation. If possible, Db2 places the copies on separate volumes.

Utilities that support backup and recovery

The following utilities are commonly used for backup and recovery:

- COPY, QUIESCE, MERGECOPY, and BACKUP SYSTEM for backup
- RECOVER, REBUILD INDEX, REPORT, and RESTORE SYSTEM for recovery

In general, you use these utilities to prepare for recovery and to restore data. Each utility plays a role in the backup and recovery process.

COPY

The COPY utility creates up to four image copies of table spaces, indexes, and data sets.

The two types of image copies are as follows:

- *Full image copy*: A copy of all pages in a table space, partition, data set, or index space.
- *Incremental image copy*: A copy of only the table space pages that have changed since the last use of the COPY utility.

While COPY is running, you can use a SHRLEVEL option to control whether other programs can access or update the table space or index.

- SHRLEVEL REFERENCE gives other programs read-only access.
- SHRLEVEL CHANGE allows other programs to change the table space or index space.

In general, the more often that you make image copies, the less time that recovery takes. However, if you make frequent image copies, you also spend more time making copies.

The RECOVER utility uses these copies when recovering a table space or index space to the most recent point in time or to a previous point in time. The catalog table SYSIBM.SYSCOPY records information about image copies.

QUIESCE
The QUIESCE utility establishes a single point of consistency, called a *quiesce point*, for one or more page sets. To establish regular recovery points for subsequent point-in-time recovery, you must run QUIESCE frequently between regular executions of COPY.

MERGECOPY
The MERGECOPY utility merges image copies that the COPY utility produced or inline copies that the LOAD or REORG utilities produced. MERGECOPY can merge several incremental copies of a table space to make one incremental copy. It can also merge incremental copies with a full image copy to make a new full image copy.

BACKUP SYSTEM
The online BACKUP SYSTEM utility invokes z/OS DFSMShsm (Version 1 Release 5 or above). BACKUP SYSTEM copies the volumes on which the Db2 data and the Db2 log information reside for a non-data sharing Db2 subsystem or a Db2 data sharing group.

RECOVER
The RECOVER utility recovers data to the current state or to a previous point in time by restoring a copy, and then by applying log records.

REBUILD INDEX
The REBUILD INDEX utility reconstructs indexes from the table that they reference.

REPORT
The REPORT utility provides information that is needed to recover a table space, an index, or a table space and all of its indexes. You can also use the REPORT utility to obtain recovery information about the catalog.

RESTORE SYSTEM
The online RESTORE SYSTEM utility invokes z/OS DFSMShsm (Version 1 Release 5 or above). RESTORE SYSTEM uses data that is copied by the BACKUP SYSTEM utility.

You can also use the following IBM Db2 Tools for z/OS in various backup and recovery situations:

Application Recovery tool for IMS and Db2 Databases
A tool that simplifies and coordinates the recovery of both IMS and Db2 data to a common point, reducing the time and cost of data recovery and availability.

Db2 Archive Log Accelerator
A tool that reduces the overhead that is associated with database log management to balance the increases in archive log growth.

Db2 Change Accumulation Tool for z/OS
A tool that quickly restores database objects with precision and minimal disruption, setting the scope and specificity of image copy creation through the use of control cards.

Db2 Log Analysis Tool for z/OS
A tool that provides you with a powerful tool to ensure high availability and complete control over data integrity. This tool allows you to monitor data changes by automatically building reports of changes that are made to database tables.

Db2 Object Restore for z/OS
A tool that enables you to recover valuable data assets by quickly restoring dropped objects without down time, even if they no longer exist in the Db2 catalog. Such dropped objects might include databases, table spaces, tables, indexes, data, and table authorizations.

Related information

Operation and recovery (Db2 Administration Guide)

Db2 restart

A key to the perception of high availability is getting the Db2 subsystem restarted quickly after an unplanned outage.

- Some restart processing can occur concurrently with new work. Also, you can choose to postpone some processing.
- During a restart, Db2 applies data changes from its log that was not written at the time of failure. Some of this process can be run in parallel.
- You can register Db2 to the Automatic Restart Manager of OS/390®. This facility automatically restarts Db2 should it go down as a result of a failure.

Related information

Operation and recovery (Db2 Administration Guide)

Regular backups and data checks

Scheduling backups and data checks on a regular basis is important. Your site must have a schedule in place to periodically check data for damage and consistency. You must also check storage structures for efficient use, and gather information to tune your Db2 subsystem for optimal performance.

Specifically, schedule the following activities:

- Take frequent backups to prepare for potential recovery situations. You must regularly take full or incremental image copies of Db2 data structures and Db2 subsystem structures.
- Use the CHECK utility periodically or after a *conditional restart* or recovery to ensure data consistency and to ensure that data is not damaged. A conditional restart lets you skip a portion of log processing during Db2 restart.
  - The CHECK DATA utility checks table spaces for violations of referential and check constraints and reports that information. You must run CHECK DATA after a conditional restart or a point-in-time recovery on all table spaces in which parent and dependent tables might not be synchronized. You can also run CHECK DATA to:
    - Check the consistency between a LOB table space or XML table space and its base table space.
    - Check the validity of the contents of an XML table space.
  - The CHECK INDEX utility tests whether indexes are consistent with the data that they index. You must run CHECK INDEX after a conditional restart or a point-in-time recovery on all table spaces with indexes that might not be consistent with the data. You must also use CHECK INDEX before running CHECK DATA to ensure that the indexes that CHECK DATA uses are valid.
- Run the REORG utility when data needs to be organized and balanced in index spaces and table spaces.
- Use the RUNSTATS utility to gather statistics about Db2 objects. Db2 uses these statistics to select the most efficient access path to data.

Related concepts

Guidelines for data reorganization
You must consider several factors before you reorganize your data.

Related information

Operation and recovery (Db2 Administration Guide)

Control of database changes and data consistency

Before you can fully understand how backup and recovery works, you need to be familiar with how Db2 keeps data consistent as changes to data occur.

The processes that ensure data consistency include commit and rollback operations and locks. This information provides an overview of how commit and rollback operations achieve a point of data consistency, and explains how Db2 maintains consistency when data is exchanged between servers.

Related information

Operation and recovery (Db2 Administration Guide)

Commit and rollback of transactions

At any time, an application process might consist of a single transaction. However the life of an application process can involve many transactions as a result of commit or rollback operations.

A transaction begins when data is read or written. A transaction ends with a COMMIT or ROLLBACK statement or with the end of an application process.

- The COMMIT statement commits the database changes that were made during the current transaction, making the changes permanent.

  Db2 holds or releases locks that are acquired on behalf of an application process, depending on the isolation level in use and the cause of the lock.
- The ROLLBACK statement backs out, or cancels, the database changes that are made by the current transaction and restores changed data to the state before the transaction began.

The initiation and termination of a transaction define *points of consistency* within an application process. A point of consistency is a time when all recoverable data that an application program accesses is consistent with other data. The following figure illustrates these concepts.

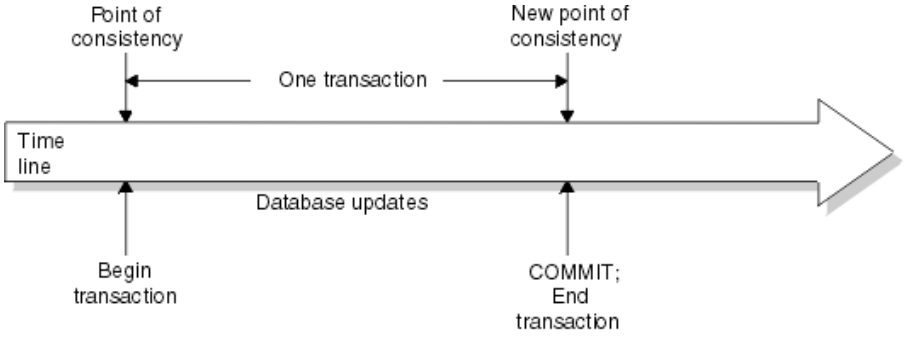


Figure 46. A transaction with a commit operation

When a rollback operation is successful, Db2 backs out uncommitted changes to restore the data consistency that existed when the unit of work was initiated. That is, Db2 *undoes* the work, as shown in the following figure. If the transaction fails, the rollback operations begins.

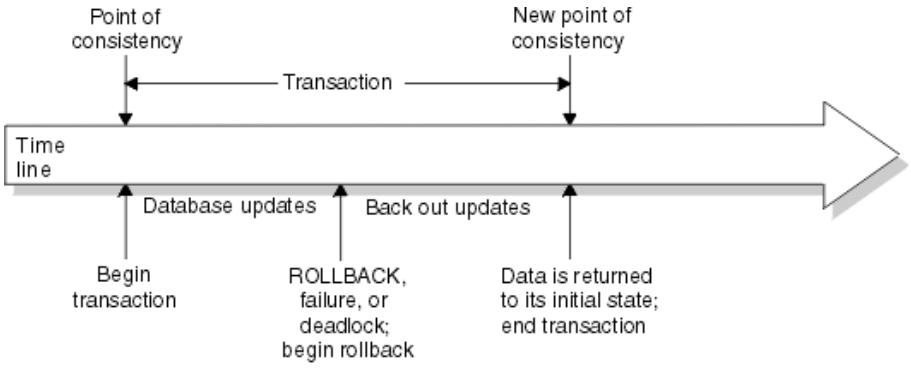


Figure 47. Rolling back changes from a transaction

An alternative to cancelling a transaction is to roll back changes to a savepoint. A *savepoint* is a named entity that represents the state of data at a particular point in time during a transaction. You can use the ROLLBACK statement to back out changes only to a savepoint within the transaction without ending the transaction.

Savepoint support simplifies the coding of application logic to control the treatment of a collection of SQL statements within a transaction. Your application can set a savepoint within a transaction. Without affecting the overall outcome of the transaction, application logic can undo the data changes that were

made since the application set the savepoint. The use of savepoints makes coding applications more efficient because you don't need to include contingency and what-if logic in your applications.

Now that you understand the commit and rollback process, the need for frequent commits in your program becomes apparent.

Related concepts

Application processes and transactions
An application process involves running one or more programs. Different application processes might involve running different programs or running the same program at different times. When an application interacts with a Db2 database, a transaction begins.

Related information

Operation and recovery (Db2 Administration Guide)

Coordinated updates for consistency between servers

In a distributed system, a transaction might occur at more than one server. To ensure data consistency, each subsystem that participates in a single transaction must coordinate update operations. Transactions must be either committed or backed out.

Db2 uses a two-phase commit process with a wide variety of resources, such as relational databases that are accessed through DRDA. Db2 support for two-phase commit can also be used from a number of different application environments. Db2 can work with other z/OS transaction management environments, such as IMS and CICS, and in UNIX environments, Microsoft Windows applications, and WebSphere Application Server.

With two-phase commit, you can update a Db2 table and data in non-Db2 databases within the same transaction. The process is under the control of one of the subsystems, called the *coordinator*. The other systems that are involved are the *participants*. For example, IMS, CICS, or RRS is always the coordinator in interactions with Db2, and Db2 is always the participant. Db2 is always the coordinator in interactions with TSO and, in that case, completely controls the commit process. In interactions with other DBMSs, including other Db2 subsystems, your local Db2 subsystems can be either the coordinator or a participant.

Related concepts

How updates are coordinated across distributed systems
Various products are available to work with Db2 to coordinate updates across a distributed transaction. Db2 coordinates updates at servers that support different types of connections.

Related information

Operation and recovery (Db2 Administration Guide)

Events in the recovery process

Db2 can recover a page set by using a backup copy.

The Db2 recovery log contains a record of all changes that were made to the page set. If the data needs to be recovered, Db2 restores the backup copy and applies the log changes to the page set from the point of the backup copy.

To recover a page set, the RECOVER utility typically uses these items:

- A full image copy; which is a complete copy of the page set.
- For table spaces only, any later incremental image copies that summarizes all changes that were made to the table space since the time that the previous image copy was made.
- All log records for the page set that were created since the most recent image copy.

The following figure shows an overview of a recovery process that includes one complete cycle of image copies.

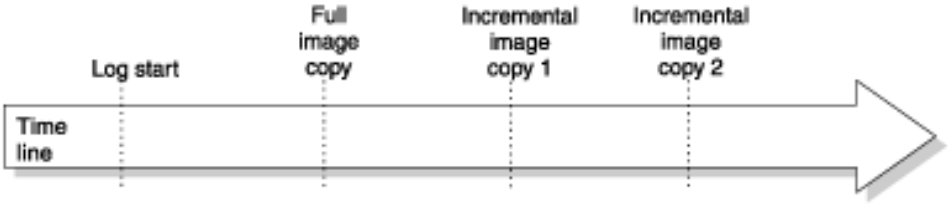


Figure 48. Overview of Db2 recovery

The SYSIBM.SYSCOPY catalog table can record many complete cycles. The RECOVER utility uses information in the SYSIBM.SYSCOPY catalog table for the following purposes:

- To restore the page set with data in the most recent full image copy
- For table spaces only, to apply all the changes that are summarized in any later incremental image copies
- To apply all changes to the page set that are registered in the log, beginning with the most recent image copy

If the log was damaged or discarded, or if data was changed erroneously and then committed, you can recover to a particular point in time. This type of recovery limits the range of log records that the RECOVER utility is to apply.

Related information

Operation and recovery (Db2 Administration Guide)

Optimization of availability during backup and recovery

Because backup and recovery affects data availability, you should understand the implications of various activities, including running utilities, logging, archiving, disaster recovery, and Db2 restart.

Running utilities

- To reduce recovery time, you can use the RECOVER utility to recover a list of objects in parallel.
- To reduce copy time, you use the COPY utility to make image copies of a list of objects in parallel.

Logging

- To speed recovery, place active or archive logs on disk. If you have enough space, use more active logs and larger active logs.
- Make the buffer pools and the log buffers large enough to be efficient.
- If Db2 is forced to a single mode of operation for the bootstrap data set or logs, you can usually restore dual operation while Db2 continues to run. Dual active logging improves recovery capability in the event of a disk failure. You can place copies of the active log data sets and the bootstrap data sets on different disk units.
- If an I/O error occurs when Db2 is writing to the log, Db2 continues to operate. If the error is on the active log, Db2 moves to the next data set. If the error is on the archive log, Db2 dynamically allocates another archive log data set.

Restart

Many recovery processes involve restarting Db2. You can minimize Db2 restart time after an outage to get the Db2 subsystem up and running quickly.

- For non-data-sharing systems, you can limit backout activity during Db2 restart. You can postpone the backout of long-running transactions until after the Db2 subsystem is operational.
- Some restart processing can occur concurrently with new work. You can choose to postpone some processing to get Db2 running more quickly.
- During a restart, Db2 applies data changes from the log. This technique ensures that data changes are not lost, even if some data was not written at the time of the failure. Some of the work to apply log changes can run in parallel.

- You can register Db2 with the Automatic Restart Manager of z/OS. This facility automatically restarts Db2 in the event of a failure.

Archiving
If you archive to tape, be sure that you have enough tape drives. Db2 then does not need to wait for an available drive on which to mount an archive tape during recovery.

Recommendation: For fast recovery, keep at least 24 hours of logs in the active logs, and keep as many archive logs as possible (48 hours of logs, for example) on disk. Archiving to disk and letting HSM (Hierarchical Storage Management) migrate to tape is a good practice.

Disaster recovery
In the case of a disaster that causes a complete shutdown of your local Db2 subsystem, your site needs to ensure that documented procedures are available for disaster recovery. For example, a procedure for off-site recovery keeps your site prepared.

Optionally, you can use DFSMShsm to automatically manage space and data availability among storage devices in your system. For example, DFSMShsm manages disk space by moving data sets that have not been used recently to less expensive storage. DFSMShsm makes data available for recovery by automatically copying new or changed data sets to tape or disk.

Related information
Operation and recovery (Db2 Administration Guide)

Data encoding basics

Most Db2 for z/OS subsystems process data from multiple encoding schemes or operate in a distributed environment where different applications must communicate with each other. In this environment, a basic understanding of code pages and CCSIDs can help you avoid certain types of performance problems or, worse yet, data corruption.

What is a code page and a CCSID?

A *code page* is a map that tells a computer which hexadecimal numbers represent which characters. Another similar term is CCSID. A *CCSID* is a number that identifies an implementation of a code page at a particular point in time. Although these terms are sometimes used interchangeably, they are different. If you are not familiar with these terms and the difference between them, read Code pages and CCSIDs (Db2 Internationalization Guide).

Db2 for z/OS uses CCSID values, not code page numbers. To indicate the encoding value of some data, specify the CCSID. Db2 uses this CCSID value to interpret the data. If you do not specify the correct CCSID, the data can be misinterpreted or corrupted by the receiving program.

Each CCSID belongs to an encoding scheme. The major encoding schemes are EBCDIC, ASCII, and Unicode. To learn more about encoding schemes and the relationship between CCSIDs and encoding schemes, read Encoding schemes (Db2 Internationalization Guide).

When data is passed between different systems or applications, it might need to be converted from one CCSID to another CCSID. That process is called *character conversion*. In some cases, character conversion can happen within the same system. Ideally, you want to avoid character conversions whenever possible because they can degrade performance.

Which CCSIDs are my Db2 subsystems using?

Each Db2 for z/OS subsystem has three default CCSID sets, one for each encoding scheme: ASCII, EBCDIC, and Unicode. Your system administrator can set the value for the ASCII CCSID set and the EBCDIC CCSID set. The Unicode CCSID set is predefined and cannot be changed.

Each subsystem also has two default encoding schemes, one for the subsystem and one for applications. The subsystem value determines the default encoding scheme of Db2 objects. The application value determines the default encoding scheme of the data that is passed to Db2 by applications.

For more information about the default CCSID and encoding scheme values, including where they are specified, see Subsystem CCSIDs and encoding schemes (Db2 Internationalization Guide).

To determine the values that are defined for your subsystem, either ask your system programmer or follow the procedure in Determining current subsystem CCSID and encoding scheme values (Db2 Internationalization Guide).

The default CCSID and encoding scheme values affect you in a couple of ways:

- They potentially affect how you create database objects. Consider the following examples:
  - You create a table and specify CCSID EBCDIC in the CREATE TABLE statement. Db2 defines that table with the default EBCDIC CCSID value. Suppose that value on your subsystem is 500. Db2 stores all data in that table as encoded in CCSID 500.
  - You create a table and do not define a CCSID. In this case, Db2 uses the default subsystem encoding scheme. If that value is EBCDIC and the default EBCDIC CCSID is 500, the table is again created with CCSID 500.

  If your data is encoded in CCSID 500, both of these examples work fine. However, if your data is encoded in another CCSID, such as CCSID 857, both of these examples result in data corruption; Db2 is using the wrong code page to interpret the data.
- They affect how you write applications that pass variables to Db2. Suppose that your application passes data to Db2 but does not tell Db2 the CCSID of the data that it is passing. Db2 then assumes that the data is in the subsystem default application encoding scheme. Again, suppose that value is EBCDIC and suppose that this time, your default EBCDIC CCSID is 37. Db2 interprets the data that is being passed as being encoded in CCSID 37. If the data is encoded in CCSID 37, this scenario works fine. However, if the data is encoded in another CCSID, such as CCSID 860, Db2 misinterprets this data from the application, and it is now corrupted.

Which CCSIDs are my other data sources using?

Knowing which CCSIDs all of your data sources are using is critical to making sure that the data is interpreted correctly and with minimal conversions.

Consider the example in which you created a table with CCSID 500. Now suppose that you are loading data into that table, but the data that you are loading is encoded in CCSID 37. Assume that you use the LOAD utility and specify CCSID 37 for the input data. The data must be converted from CCSID 37 to CCSID 500. Although these CCSIDs are similar, some characters will need to be converted. The greater the number of characters that must be converted, the bigger the impact is on performance.

Now consider the example in which your application does not specify the CCSID of the data that it is passing to Db2. Again, suppose that the default application encoding scheme is EBCDIC, and the default EBCDIC CCSID is 37. Db2 assumes that the data it receives is in CCSID 37. However, suppose that the data that you are passing to Db2 is actually in another CCSID. For example, suppose that you are passing Japanese characters, encoded in CCSID 281. Db2 is reading the data as if it is encoded in CCSID 37; therefore, the data is corrupted.

Consider a more subtle example. Suppose that the data that you are passing is in CCSID 1140. Only one code point is different between CCSID 37 and CCSID 1140: the character at code point X'9F'. If the data that is being passed to Db2 does not contain this character, you might not even notice a problem. Suppose, however, that one day this character is passed to Db2. The application passes € but Db2 interprets it using CCSID 37, which maps to ¤. Your data now has one corrupted character, and it might take some time to discover the error.

These examples illustrate the importance of knowing the CCSID of all of the data that you are working with. The following resources can help you find the CCSID values of some of your data sources:

- To find the CCSID for some of the common data sources, use the table in Finding the CCSID values of your data sources (Db2 Internationalization Guide).
- To find the CCSID of Db2 objects, use the instructions in Determining the CCSID of Db2 data (Db2 Internationalization Guide).

Where do I specify CCSIDs?

You can specify CCSIDs for objects and applications:

- For objects, you specify CCSIDs when you create them. See Specifying object CCSIDs (Db2 Internationalization Guide) for more information.
- For Db2 for z/OS applications, you can specify a CCSID for the program source code and a CCSID for the variables. The way that you specify the CCSID for the source code depends on whether you use the Db2 precompiler or the Db2 coprocessor and the language that you are using. For the application data (the host variables and parameter markers), generally you use the ENCODING bind option. For more information about the process and all of the options available, seeSpecifying a CCSID for your application (Db2 Internationalization Guide).

What is Unicode?

You can think of Unicode as one giant code page that includes all characters. For a detailed explanation, see Unicode (Db2 Internationalization Guide) and Why the Unicode standard? (z/OS: Unicode Services User's Guide and Reference).

Unicode has a couple of formats that Db2 for z/OS supports: UTF-8 and UTF-16. These formats are explained in UTFs (Db2 Internationalization Guide) and Deciding whether to store data as UTF-8 or UTF-16 (Db2 Internationalization Guide).

Other learning resources

See Introduction to character conversion (Db2 Internationalization Guide) for a comprehensive explanation of CCSIDs, Unicode, and international data, including how to specify the sort sequence of a language and how to perform culturally correct case conversions.

For more detailed training about Unicode, use the following resources:

- Course: "How to Deal with Unicode and Multiple CCSIDs." You can find links to this course information from the Db2 for z/OS Database Administrator training path.
- Unicode Tutorials and Overviews (on Unicode Consortium website)

Introduction to troubleshooting problems in Db2 for z/OS

If you are new to Db2 for z/OS, you should learn about the tools and techniques that are available to help you troubleshoot and resolve problems.

These tools and techniques include:

- "Messages and codes" on page 201
- "Getting information from the Db2 catalog" on page 202
- "Db2 Diagnosis Guide and Reference" on page 203
- "Monitoring threads" on page 203
- "Traces" on page 203
- "Dumps" on page 204
- "Troubleshooting specific types of problems" on page 204
- "Fixes" on page 205
- "Contacting IBM Support" on page 205

Messages and codes

One of the first tools that you typically use to troubleshoot a problem are the messages and codes that Db2 issues:

Error, warning, and informational messages

Db2 for z/OS message numbers begin with the prefix DSN. The message number is followed by some explanatory text. Depending on the message and the situation, messages are either displayed on the console, returned in the job output, or passed back to the application. Messages do not always indicate a problem. Some messages merely provide information about the state of the system and do not require a user response.

For more information about the various parts of a message number, see How to interpret message numbers (Db2 Messages).

If you receive messages from other IBM products (messages that do not begin with DSN), use the following resource to find information for those messages: How to find additional information about messages (Db2 Messages).

Reason codes

Reason codes are typically presented as a value in a message that Db2 issues. Reason codes provide specific details about the circumstances of a particular instance of a message.

For information about the various parts of a reason code number, see How to interpret code numbers (Db2 Codes).

SQL codes

SQL codes are numbers that indicate the results of SQL statements. Positive and 000 SQL codes indicate that the SQL statement ran successfully. Negative SQL codes indicate an error.

The SQL code is passed to a structure called the SQL communication area (SQLCA). Your application can retrieve the SQL code value from the SQLCODE field in the SQLCA. The SQLCA also contains other information that is related to the SQL code, such as the SQLSTATE value, which also indicates whether the SQL statement was successful. Any variable values (*message tokens*) for the SQL code are also stored in the SQLCA.

For more information about SQL codes, SQLSTATE, and the SQLCA, see SQL codes (Db2 Codes).

One way to check the SQL code and other diagnostic information for SQL statements is to use the GET DIAGNOSTICS statement. See Checking the execution of SQL statements by using the GET DIAGNOSTICS statement (Db2 Application programming and SQL).

Getting information from the Db2 catalog

The Db2 *catalog* is a set of tables that contain information about the data that Db2 controls. You can use the information in the catalog to troubleshoot problems.

For example, if Db2 returns an unexpected character or a CCSID conversion error, you can look at the CCSID information in the catalog. See Collecting data for CCSID problems (Collecting data).

If you receive messages about objects that Db2 cannot find, you can check the catalog to verify that those objects really exist in the catalog or to verify that they were specified correctly.

The information in the catalog can also be useful for diagnosing authorization problems. If you are using native Db2 security, which means that Db2 checks authorization, you can use SELECT statements against the Db2 catalog to confirm that the authorizations in question exist. If you are using external Db2 security, your security administrator can run reports that show which authorizations exist.

For a general description of the catalog, see “Db2 catalog” on page 38. For a detailed list of the various catalog tables, see Db2 catalog tables (Db2 SQL).

Troubleshooting for Db2

Troubleshooting for Db2 in the Db2 for z/OS collections in IBM Knowledge Center contains detailed diagnostic information that can help you to diagnose problems in Db2.

Db2 Diagnosis Guide and Reference

The *DB2 Diagnosis Guide and Reference* contains detailed diagnostic information that can help you to diagnose problems in Db2. Typically, you will use this information only with the help of advanced Db2 for z/OS users, IBM Support, or both.

A portion of the information in the *DB2 Diagnosis Guide and Reference* is publicly available in IBM Knowledge Center, under Troubleshooting for Db2. To access the information more easily, search IBM Knowledge Center first. The *DB2 Diagnosis Guide and Reference* is a licensed publication and is therefore not publicly available. For information about how to access this publication, see Obtaining Db2 Diagnosis Guide and Reference ().

Monitoring threads

A *thread* is a structure that describes a connection made by an application and traces its progress in the Db2 subsystem. You can monitor threads by using the DISPLAY THREAD command. This command returns various information about threads, such as the connection name, the status, and the user. The information that is returned depends on the options that you specify.

For basic information about threads and the types of threads in a Db2 subsystem, see Monitoring threads (Db2 Administration Guide).

The following list gives some examples of when you might want to monitor threads:

- If you suspect that DDF or Db2 might be hung, use DISPLAY THREAD to check for suspended threads. See the examples in -DISPLAY THREAD (Db2) (Db2 Commands).
- To help identify and resolve indoubt threads, see Scenarios for resolving problems with indoubt threads (Db2 Administration Guide).
- To identify and recover from any indefinitely long wait conditions, see Recovering from an indefinite wait condition (Db2 Administration Guide).
- To determine the status of resources that you are trying to access, see the examples 00C900AB (Db2 Codes) and DSNU209I (Db2 Messages).
- To troubleshoot any messages about agents that do not exist at partner locations, see DSNL025I (Db2 Messages).
- To find a thread token so that you can cancel the thread, see -CANCEL THREAD (Db2) (Db2 Commands).

See Monitoring threads (Db2 Administration Guide) for detailed instructions. Be aware that the techniques for diagnosing problems with distributed threads are different from diagnosing problems with local threads. Details about the available techniques are in Diagnostic aids for distributed data (Diagnosing Db2 problems) and Diagnostic aids for single systems and data sharing (Diagnosing Db2 problems).

Traces

A *trace* is a log of monitoring, auditing, performance, accounting, statistics, or serviceability data that is captured over a given time period. In Db2 for z/OS, the trace facility is called the *instrumentation facility component (IFC)*. When you run a trace, the IFC gathers information about certain Db2 events. Each piece of information that is gathered has a numeric identifier. These identifiers are called *IFCIDs*. For example, IFCID 0305 records information that is related to defining, removing, and enforcing a table check constraint.

There are several types of Db2 for z/OS traces. A *trace type* is the type of information that IFC gathers, such as performance information or statistics information. Within each trace type is a number of trace classes. A *trace class* is a number that identifies a group of one or more IFCIDs.

For example, if you start performance trace class 1, the following IFCIDs are activated:

0001, 0002, 0031, 0042, 0043, 0076-0079, 0102, 0103, 0105-0107, 0153

Similarly, if you start statistics trace class 1, the following IFCIDs are activated:

0001, 0002, 0105, 0106, 0202, 0225

Notice that an IFCID can belong to more than one trace class.

For a description of each trace type, including a list of its classes and corresponding IFCIDs, see the following information:

- Statistics trace (Db2 Performance)
- Accounting trace (Db2 Performance)
- Audit trace (Db2 Performance)
- Performance trace (Db2 Performance)
- Monitor trace (Db2 Performance)

For a description of all of the IFCIDs, view the contents of *prefix*.SDSNIVPD(DSNWMSGS). For more information about DSNWMSGS and all of the information that it contains, see Trace field descriptions (Db2 Performance).

To run a trace, use the Db2 START TRACE command. (This command invokes the IFC to run a trace.) See -START TRACE (Db2) (Db2 Commands).

For more information about other commands that you can use to control traces, see Controlling the Db2 trace (Db2 Administration Guide).

Each trace produces a number of trace records. Generally, you will use tools such as Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS to interpret these trace records. However, you can analyze traces manually. For a description of Db2 trace output, see Db2 trace output (Db2 Performance).



Attention: Be careful about running too many traces because of the processing cost. See Minimizing the processing cost of Db2 traces (Db2 Performance). Also, you should limit the amount of trace data that you request so that you do not impact system performance. See Minimizing the volume of Db2 trace data (Db2 Performance).

Dumps

A *dump* is a copy of the contents of memory at a particular point in time. Dumps are a useful tool for diagnosing and debugging problems in Db2. Often, IBM Support will request that you generate a dump to help them understand the state of your system when a problem occurred.

For more information about the different types of dumps, see Dumps (z/OS basic skills).

Additional guidance about the types of Db2 dumps and analysis is in the Printing and analyzing dumps (Diagnosing Db2 problems).

For instructions for generating a Db2 SVC dump, see Requesting Db2 SVC dumps (Collecting data).

If you concatenate the Db2 load library to the Interactive Program Control System (IPCS) startup procedures, you can analyze dumps yourself. For information about the IPCS commands to issue, see Format dumps by using IPCS options (Diagnosing Db2 problems)

You can also use the DSN1SDMP stand-alone utility to force a dump when certain trace events occur. For example, you might want to generate a dump for a particular negative SQL code. For an example of this situation, see example 4 in Sample DSN1SDMP control statements (Db2 Utilities). For more information about DSN1SDMP, see DSN1SDMP (Db2 Utilities).

Troubleshooting specific types of problems

You can use the following information to troubleshoot specific types of problems:

- Debugging an application program (Db2 Application programming and SQL)
- Troubleshooting Db2 stored procedure problems (Db2 Administration Guide)
- Debugging CCSID and Unicode problems (Db2 Internationalization Guide)
- Problem diagnosis (Db2 Programming for ODBC) (ODBC)
- Investigating Db2 performance problems (Db2 Performance)

- Troubleshooting the administrative task scheduler (Db2 Administration Guide)
- Investigating access path problems (Db2 Performance)
- Steps for diagnosing a system hang (z/OS basic skills)

If you recently migrated to a new version of Db2 for z/OS, check for any release incompatibilities that might be the source of the problem. See Incompatible changes for migration to Db2 11 (Db2 for z/OS What's New?).

Fixes

If you suspect that the problem is with Db2 for z/OS, ask your system programmer to help you search for any available fixes. For Db2 for z/OS, a fix is called a *PTF (program temporary fix)*. Potentially, someone else has already experienced the same problem and a PTF is available or will be available soon. In some cases, interim workarounds might be provided.

You can also search for *authorized program analysis reports (APARs)*. An APAR is a formal problem report of a suspected defect in the product. An APAR can contain known workarounds and an indication of whether a PTF is planned.

You can search for both PTFs and APARs in the IBM Support Portal: Db2 for z/OS. For a list of keywords that you can use to optimize your search results when searching APAR text, see Db2 for z/OS Keywords for searching APAR text.

For instructions for obtaining PTFs, see Getting fixes.

Contacting IBM Support

If you cannot solve the problem, contact IBM Support by using the following procedure: Contacting IBM Support about Db2 problems ().

When you contact IBM Support, you can expect to send certain data, such as the following information:

- A complete SQLCA for a failing SQL statement
- A complete dump for a Db2 abend
- *Service SQL documentation* as described in Collecting service SQL documentation ().
- SMF or RMF data for Db2 system performance problems
- Precompiler or compiler output for SQL issues that are related to program preparation
- Full Db2 error message text if a message is issued
- LOGREC and SYSLOG for abend or environmental issues

For instructions for gathering this data, see Collecting diagnostic data (Collecting data).

Related information
Db2 for z/OS training paths

Chapter 4. Db2 performance management

Managing the performance of a Db2 subsystem involves understanding a wide range of system components. You need to understand the performance of those components, how to monitor the components, and how to identify problem areas.

System resources, database design, and query performance are among the many performance issues to consider, and each of these factors influences the others. For example, a well-designed query does not run efficiently if system resources are not available when it needs to run.

To manage Db2 performance, you need to establish performance objectives and determine whether objects, resources, and processes are meeting your performance expectations. Tips and guidelines help you tune your Db2 subsystem to improve performance. Several tools are available to make performance analysis easier for you.

Related information

Db2 12 for z/OS Performance Topics (IBM Redbooks)

Db2 11 for z/OS Performance Topics (IBM Redbooks)

Subsystem and Transaction Monitoring and Tuning with Db2 11 for z/OS (IBM Redbooks)

DB2 10 for z/OS Performance Topics (IBM Redbooks)

Initial steps for performance management

The first step in managing Db2 performance is understanding performance issues. You need to know how to recognize different types of performance problems and to know what tools are available to help you solve them.

Performance objectives

Establishing performance objectives can help you make good choices as you work with Db2. Although performance objectives vary for every business, how your site defines good Db2 performance depends on data processing needs and priorities.

In all cases, performance objectives must be realistic, understandable, and measurable. Typical objectives include values for:

- Acceptable response time (a duration within which some percentage of all applications have completed)
- Average throughput (the total number of transactions or queries that complete within a given time)
- System availability, including mean time to failure and the durations of downtimes

Objectives such as these define the workload for the system and determine the requirements for resources, which include processor speed, amount of storage, additional software, and so on.

Example: An objective might be that 90% of all response times on a local network during a prime shift are under 2 seconds. Another objective might be that the average response time does not exceed 6 seconds, even during peak periods. (For remote networks, response times are substantially higher.)

Often, though, available resources limit the maximum acceptable workload, which requires that you revise the objectives.

Related tasks

Setting performance objectives and defining your workloads (Db2 Performance)

© Copyright IBM Corp. 2001, 2020

Application design for performance

Designing the database and applications to be as efficient as possible is an important first step to good system and application performance. As you code applications, consider performance objectives in your application design.

Tip: You can use tools such as IBM Data Server Manager and related Db2 Tools for z/OS solutions to simplify this task.

- "IBM Data Server Manager" on page 305
- Db2 Tools for z/OS

Some factors that affect the performance of applications include how the program uses host variables and what bind options you choose. In turn, those factors affect how long Db2 takes to determine an access path for the SQL statements in the application.

Later in this information you can read about locking and concurrency, including recommendations for database and application design that improve performance.

After you run an application, you need to decide if it meets your performance objectives. You might need to test and debug the application to improve its performance.

Related concepts

Performance information for SQL application programming
Efficient applications are an important first step to good system and application performance. As you code applications that access data in Db2, consider performance objectives in your application design.

Performance objectives
Establishing performance objectives can help you make good choices as you work with Db2. Although performance objectives vary for every business, how your site defines good Db2 performance depends on data processing needs and priorities.

Setting reasonable performance objectives (Db2 Performance)

Investigating SQL performance by using EXPLAIN (Db2 Performance)

Interpreting data access by using EXPLAIN (Db2 Performance)

Related tasks

Programming applications for performance (Db2 Performance)

Improving concurrency (Db2 Performance)

Programming for concurrency (Db2 Performance)

Origin of performance problems

After running an application, if you determine that it does not meet your performance objectives, you need to determine the origin of the problem. This information describes how you identify performance problems and what tools can help you.

To identify a performance problem, you begin by looking at the overall system before you decide that you have a problem in Db2. In general, look closely to see why application processes are progressing slowly or why a given resource is being heavily used.

Within Db2, the performance problem is usually either poor response time or an unexpected and unexplained high use of resources. Check factors such as total processor usage, disk activity, and memory usage.

First, get a picture of task activity, from classes 1, 2, and 3 of the accounting trace. Db2 provides a *trace facility* that lets you monitor and collect detailed information about Db2, including performance and statistical information. Then, focus on specific activities, such as specific application processes or a specific time interval. You might see problems such as these:

- Slow response time. You can collect detailed information about a single slow task, a problem that can occur for several reasons. For example, users might be trying to do too much work with certain applications, and the system simply cannot do all the work that they want done.

- Real storage constraints. Applications progress more slowly than expected because of paging interrupts. The constraints result in delays between successive requests that are recorded in the Db2 trace.

If you identify a performance problem in Db2, you can look at specific reports. Reports give you information about:

- Whether applications are able to read from buffer pools rather than from disk
- Whether and how long applications must wait to write to disk or wait for a lock
- Whether applications are using more than the usual amount of resources

Db2 also provides several tools that help you analyze performance.

Related concepts

The role of buffer pools in caching data
Buffer pools are a key element of Db2 performance, and help you to avoid delays when retrieving data.

Ways to improve performance for multiple users
Locking ensures the integrity and accuracy of data. However, locking is sometimes too restrictive and can cause slow performance and poor concurrency. Understanding how locking works can help you make good decisions for performance. Locking is critical to performance, and also provides recommendations for promoting concurrency.

Tools for performance analysis
You can use workstation and browser-based tools to simplify performance analysis for Db2 for z/OS.

Db2 trace output (Db2 Performance)
Db2 trace (Db2 Performance)

Tools for performance analysis

You can use workstation and browser-based tools to simplify performance analysis for Db2 for z/OS.

For example, the following tools simplify performance analysis for Db2 for z/OS:

- “IBM Data Server Manager” on page 305
- “IBM Data Studio” on page 307
- Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS
- Introduction to Buffer Pool Analyzer (Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS)
- Db2 SQL Performance Analyzer for z/OS
- Db2 Query Monitor for z/OS

You can also use the Db2 EXPLAIN facility to analyze query access paths. For more information, see Investigating SQL performance by using EXPLAIN (Db2 Performance).

Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS

Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS integrates performance monitoring, reporting, buffer pool analysis, and a performance warehouse function into one tool. It provides a single-system overview that monitors all subsystems and instances across many different platforms in a consistent way.

Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS includes the function of Db2 Performance Monitor on z/OS. It provides the following features:

- Combined information from EXPLAIN and from the Db2 catalog.
- Displays of access paths, indexes, tables, table spaces, plans, packages, DBRMs, host variable definitions, ordering, table access sequences, join sequences, and lock types.
- An immediate "snapshot" view of Db2 for z/OS activities that the online monitor provides. The monitor allows for exception processing while the system is operational.

Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS offerings support Db2 for z/OS on IBM Z, and Db2 for Linux, UNIX, and Windows, HP-UX, Sun Solaris, and IBM AIX®).

Related concepts

The role of buffer pools in caching data

Buffer pools are a key element of Db2 performance, and help you to avoid delays when retrieving data.

Related reference

Db2 Tools for z/OS

Related information

Db2 Tools for z/OS Product Documentation

Ways to move data efficiently through the system

As data progresses through a Db2 subsystem, it moves from disk to memory and to the user or to applications. You need to tune the system resources and objects such as buffer pools, table spaces, and indexes, that contain data to keep the flow of data efficient.

The role of buffer pools in caching data

Buffer pools are a key element of Db2 performance, and help you to avoid delays when retrieving data.

Db2 can retrieve a page from a buffer pool faster than it can from disk. When data is already in a buffer, an application program avoids the delay of waiting for Db2 to retrieve the data from disk.

You can specify that Db2 uses up to 50 buffer pools that contain 4 KB pages. You can also specify up to 10 buffer pools each that contain 8 KB, 16 KB, and 32 KB pages.

The following figure shows buffer pools with 4 KB and 8 KB pages. The number of pages that a buffer pool contains depends on the size of the buffer pool.

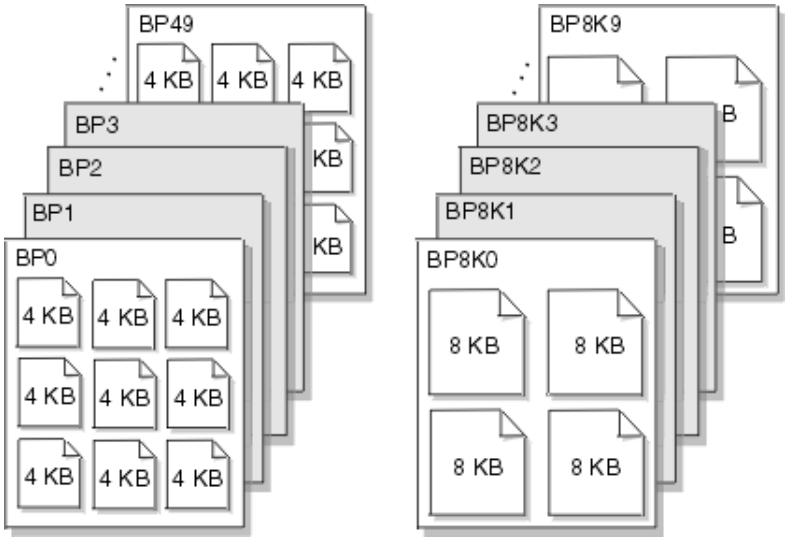


Figure 49. Buffer pools with 4 KB and 8 KB pages

At any time, pages in a virtual buffer pool can be in-use, updated, or available:

- *In-use pages* are currently being read or updated. The data that they contain is available for use by other applications.
- *Updated pages*contain changed data that is not yet written to disk.
- *Available pages* are ready for use. An incoming page of new data can overwrite available pages.

To avoid disk I/O, you can use updated and available pages that contain data.

When data in the buffer changes, that data must eventually be written back to disk. Because Db2 does not need to write the data to disk right away, the data can remain in the buffer pool for other uses. The data remains in the buffer until Db2 decides to use the space for another page. Until that time, applications can read or change the data without a disk I/O operation.

The key factor that affects the performance of buffer pools is their size. The method that Db2 uses to access buffer pools also affects performance.

Buffer pool size

The size of buffer pools is critical to the performance characteristics of an application or a group of applications that access data in those buffer pools.

By tuning your buffer pools, you can improve the response time and throughput for your applications, and provide optimum use of resources. For example, applications that process online transactions are more likely to need large buffer pools because they often must reaccess the same data. In those cases, storing large amounts of data in a buffer pool enables applications to access data more efficiently.

By making buffer pools as large as possible, you can achieve the following benefits:

- Fewer I/O operations result, which means faster access to your data.
- I/O contention is reduced for the most frequently used tables and indexes.
- Sort speed is increased because of the reduction in I/O contention for work files.

You can use the ALTER BUFFERPOOL command to change the size and other characteristics of a buffer pool at any time while Db2 is running. Use the DISPLAY BUFFERPOOL and ALTER BUFFERPOOL commands to gather buffer pool information and change buffer pool sizes.

Db2 Buffer Pool Analyzer for z/OS helps database administrators manage buffer pools more efficiently by providing information about current buffer pool behavior and by using simulation to anticipate future behavior. Using this tool, you can take advantage of these features:

- Collection of data about virtual buffer pool activity
- Comprehensive reporting of the buffer pool activity
- Simulated buffer pool usage
- Reports and simulation results
- Expert analysis that is available through an easy-to-use wizard

Db2 Buffer Pool Analyzer capabilities are included in Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS Db2 Performance Expert.

Efficient page access

Db2 determines when to use a method called *sequential prefetch* to read data pages faster. With sequential prefetch, Db2 determines in advance that a set of data pages is about to be used. Db2 then reads the set of pages into a buffer with a single I/O operation. The prefetch method is always used for table space scans and is sometimes used for index scans. Prefetching is processed concurrently with other application I/O operations.

In addition to a predetermined sequential prefetch, Db2 also supports dynamic prefetch. A dynamic prefetch is a more robust and flexible method that is based on sequential detection.

Related concepts

How z/OS uses physical and virtual storage (z/OS basic skills)

What is paging? (z/OS Basic Skills)

Buffer pools

Buffer pools are areas of virtual storage that temporarily store pages of table spaces or indexes.

Read operations and prefetch I/O (Db2 Performance)

Tools for performance analysis

You can use workstation and browser-based tools to simplify performance analysis for Db2 for z/OS.

Related tasks

Tuning database buffer pools (Db2 Performance)

Related reference

-ALTER BUFFERPOOL (Db2) (Db2 Commands)

-DISPLAY BUFFERPOOL (Db2) (Db2 Commands)

The effect of data compression on performance

In many cases, compressing the data in a table space significantly reduces the amount of disk space that is needed to store data. Compressing data can also help improve buffer pool performance. For example, you can store more data in a buffer pool, and Db2 can scan large amounts of data more easily.

With compressed data, performance improvements depend on the SQL workload and the amount of compression. You might see some of the following benefits:

- Higher buffer pool hit ratios. The hit ratio measures how often a page is accessed without requiring an I/O operation.
- Fewer operations in which Db2 accesses a data page.

The compression ratio that you achieve depends on the characteristics of your data. Compression can work well for large table spaces. With small table spaces, the process of compressing data can negate the space savings that compression provides.

Consider these factors when deciding whether to compress data:

- Db2 compresses data one row at a time. If Db2 determines that compressing the row yields no savings, the row is not compressed. The closer that the average row length is to the actual page size, the less efficient compression can be.
- Compressing data costs processing time. Although decompressing data costs less than compressing data, the overall cost depends on the patterns in your data.

If the compression ratio is less than 10%, compression is not beneficial and, therefore, is not recommended.

You can use the DSN1COMP utility to determine the probable effectiveness of compressing your data.

You use the COMPRESS clause of the CREATE TABLESPACE and ALTER TABLESPACE statements to compress data in a table space, data in a partition of a partitioned table space, or data in indexes. You cannot compress data in LOB table spaces.

Related concepts

Deciding whether to compress data (Db2 Performance)

Related tasks

Compressing your data (Db2 Performance)

Compressing indexes (Db2 Performance)

Related reference

DSN1COMP (Db2 Utilities)

How data organization can affect performance

To achieve optimal performance for table spaces and indexes, you need to keep data organized efficiently. The use of space and the organization of data in a table space and the associated indexes sometimes affects performance.

Related tasks

Maintaining data organization (Db2 Performance)

Use of free space in data and index storage

An important factor that affects how well your table spaces and indexes perform is the amount of available *free space*. Free space refers to the amount of space that Db2 leaves free in a table space or index when data is loaded or reorganized.

Freeing pages or portions of pages can improve performance, especially for applications that perform high-volume inserts or that update varying-length columns. When you specify a sufficient amount of free space, you trade the amount of used disk space for the performance of certain SQL statements. For example, inserting new rows into free space is faster than splitting index pages.

You use the FREEPAGE and PCTFREE clauses of the CREATE and ALTER TABLESPACE and INDEX statements to set free space values.

Related tasks

Reserving free spaces for indexes (Db2 Performance)

Reserving free space for table spaces (Db2 Performance)

Related reference

CREATE TABLESPACE (Db2 SQL)

ALTER TABLESPACE (Db2 SQL)

CREATE INDEX (Db2 SQL)

ALTER INDEX (Db2 SQL)

Guidelines for data reorganization

You must consider several factors before you reorganize your data.

You must run the REORG utility only when you determine that data needs to be reorganized. If application performance is not degraded, you might not need to reorganize data. Even when some statistics indicate that data is becoming unorganized, a REORG utility job is not always required, unless the lack of organization exceeds a specified threshold.

In the following situations, data reorganization is advisable:

Data is in REORG-pending status

When table spaces or partitions are in *REORG-pending (REORP)* status, you cannot select, insert, update, or delete data. You must reorganize table spaces or partitions when REORG-pending status imposes this restriction.

You can use the **DISPLAY DATABASE RESTRICT** command to identify the table spaces and partitions that need to be reorganized.

Data is in advisory REORG-pending status

After you change table or index definitions, consider reorganizing data to improve performance. After you change data types or column lengths by using ALTER TABLE statements, Db2 places the table space that contains the modified data in advisory REORG-pending (AREO*) status. The table space is in AREO* status because the existing data is not immediately converted to its new definition. Reorganizing the table space prevents possible performance degradation.

Recommendation: When data is in REORG-pending or AREO* status, use the REORG utility with the SCOPE PENDING option to automatically reorganize partitions. With this option, you do not need to first identify which partitions need to be reorganized or to customize the REORG control statement.

Data is skewed

When you use partitioned table spaces, you might sometimes find that data is out of balance, or skewed. When data is skewed, performance can be negatively affected because of contention for I/O and other resources. You might also have a situation in which some partitions are approaching their maximum size, and other partitions have excess space.

You can correct the skewed data by redistributing the data across partitions.

Related information:

Redistributing data in partitioned table spaces (Db2 Administration Guide)

Data is unorganized or fragmented

When data becomes unorganized or fragmented, you need to consider reorganizing your table spaces and index spaces.

You need to consider the following situations to evaluate when data reorganization is necessary:

Unused space

In simple table spaces, dropped tables use space that is not reclaimed until you reorganize the table space. Consider running REORG if the percentage of space that is occupied by rows of dropped tables is greater than 10%. The PERCDROP value in the SYSIBM.SYSTABLEPART catalog table identifies this percentage.

Page gaps

Indexes can have multiple levels of pages. An index page that contains pairs of keys and identifiers and that points directly to data is called a *leaf page*.

Deleting index keys can result in page gaps within leaf pages. Gaps can also occur when Db2 inserts an index key that does not fit onto a full page. Sometimes Db2 detects sequential inserts and splits the index pages asymmetrically to improve space usage and reduce split processing. You can improve performance even more by choosing the appropriate page size for index pages. If page gaps occur, consider running the REORG utility.

The LEAFNEAR and LEAFFAR columns of SYSIBM.SYSINDEXPART store information about the organization of physical leaf pages by indicating the number of pages that are not in an optimal position.

I/O activity

You can determine when I/O activity on a table space might be increasing. A large number (relative to previous values that you received) for the NEARINDREF or the FARINDREF option indicates an increase in I/O activity. Consider a reorganization when the sum of NEARINDREF and FARINDREF values exceeds 10%.

The NEARINDREF and FARINDREF values in the SYSIBM.SYSTABLEPART and SYSIBM.SYSTABLEPART_HIST catalog tables identify the number of reallocated rows.

Recommendation: When increased I/O activity occurs, use a non-zero value for the PCTFREE clause of the table space definition. The PCTFREE clause specifies what percentage of each page in a table space or index is left free when data is loaded or reorganized. PCTFREE is a better choice than FREEPAGE.

Clustering

You can determine if clustering is becoming degraded. Clustering becomes degraded when the rows of a table are not stored in the same order as the entries of its clustering index. A large value for the FAROFFPOSF option might indicate poor clustering. Reorganizing the table space can improve performance. Although less critical, a large value for the NEAROFFPOSF option can also indicate that reorganization might improve performance. The FAROFFPOSF and NEAROFFPOSF values in the SYSIBM.SYSINDEXPART and SYSIBM.SYSINDEXPART_HIST catalog tables identify the number of rows that are far from and near to optimal position.

REORG thresholds

You can use the RUNSTATS, REORG, REBUILD INDEX, and LOAD utilities to collect statistics that describe the fragmentation of table spaces and indexes. These statistics can help you decide when to run the REORG utility to improve performance or reclaim space.

You can set up your REORG job in accordance with threshold limits that you set for relevant statistics from the catalog. The OFFPOSLIMIT and INDREFLIMIT options specify when to run REORG on table spaces. When a REORG job runs with these options, it queries the catalog for relevant statistics. The REORG job does not occur unless one of the thresholds that you specify is exceeded. You can also specify the REPORTONLY option to produce a report that tells you whether a REORG job is recommended.

Related tasks

Maintaining data organization (Db2 Performance)

Related reference

REORG-pending status (Db2 Utilities)

REORG TABLESPACE (Db2 Utilities)

REORG INDEX (Db2 Utilities)

RUNSTATS (Db2 Utilities)
LOAD (Db2 Utilities)
SYSTABLEPART catalog table (Db2 SQL)
SYSINDEXPART catalog table (Db2 SQL)

Ways to improve performance for multiple users

Locking ensures the integrity and accuracy of data. However, locking is sometimes too restrictive and can cause slow performance and poor concurrency. Understanding how locking works can help you make good decisions for performance. Locking is critical to performance, and also provides recommendations for promoting concurrency.

Db2 uses locks on user data. The main reason for using locks is to ensure the integrity, or accuracy, of the data. Without locks, one user might be retrieving a specific data item while another user might be changing that data. The result is that the first user retrieves inaccurate data. In the Db2 for z/OS environment, which includes vast amounts of data and large numbers of users and transactions, the prospect of inaccurate data is unacceptable. Therefore, Db2 for z/OS provides comprehensive locking to ensure data integrity.

Despite the importance of data integrity, locking can sometimes be too restrictive. If an application process locks too much data, other users, utilities, and application processes must wait for the locked data. This situation results in poor concurrency. *Concurrency* is the ability of more than one application process to access the same data at essentially the same time. Db2 for z/OS handles the trade-off between concurrency and data integrity to maximize concurrency without sacrificing the integrity of the data.

Related concepts
Lock contention, suspension, timeout, and deadlock (Db2 Performance)
Related tasks
Improving concurrency (Db2 Performance)
Related reference
Db2 concurrency and locking controls (Managing Db2 Performance)

Improved performance through the use of locks

Db2 uses locks on various data objects.

Locks can be placed on rows, pages, tables, table space segments, table space partitions, entire table spaces, and databases. When an application acquires a lock, the application "holds" or "owns" the lock.

Db2 uses of the lock modes to determine whether one lock is *compatible* with another. Some lock modes do not exclude all other users. For example, assume that application process A holds a lock on a table space that process B also wants to access. Db2 requests, on behalf of process B, a lock of some particular mode. If the mode of the lock for process A permits the lock requested by process B, the modes of the two locks are said to be compatible. However, if the two locks are not compatible, process B cannot proceed. It must wait until process A releases its lock, and until all other existing incompatible locks are released.

The following lock modes provide different degrees of protection:

S lock (share)
The lock owner and any concurrent processes can read, but not change, the locked page or row. Concurrent processes can acquire S or U locks on the page or row or might read data without acquiring a page or row lock

U lock (update)
The lock owner can read, but not change, the locked page or row. Concurrent processes can acquire S-locks or might read data without acquiring a page or row lock, but no concurrent process can acquire a U-lock.

U locks reduce the chance of deadlocks when the lock owner is reading a page or row to determine whether to change it. The owner can start with the U lock and then promote the lock to an X lock to change the page or row.

X lock (exclusive)
The lock owner can read or change the locked page or row. A concurrent process cannot acquire S, U, or X locks on the page or row. However, concurrent processes, such as those processes bound with the CURRENTDATA(NO) or ISOLATION(UR) bind options or running with YES specified for the EVALUNC subsystem parameter, can read the data without acquiring a page or row lock.

The following table shows the compatibility of any two modes for page and row locks. No question of compatibility arises between page and row locks, because a partition or table space cannot use both page and row locks.

Table 25. Compatibility matrix of page lock and row lock modes

Lock mode	Share (S-lock)	Update (U-lock)	Exclusive (X-lock)
Share (S-lock)	Yes	Yes	No
Update (U-lock)	Yes	No	No
Exclusive (X-lock)	No	No	No

The share, update, and exclusive locks apply to row or page locks. These facts apply only to application processes that acquire an intent lock on the table space and the table, if the table is in a segmented table space.

When a page or row is locked, the table, partition, or table space that contains it is also locked. This *intent lock* indicates the plan that the application process has for accessing the data. In that case, the table, partition, or table space lock has one of the *intent modes*: either IS for *intent share*, IX for *intent exclusive*, or SIX for *share with intent exclusive*.

Compatibility for table space locks is slightly more complex that for page and row locks.

Locks are important for maintaining concurrency in the Db2 environment. However, locks might cause several types of *contention* situations that degrade Db2 performance, including suspension, timeout, and deadlock.

Suspension
An application encounters *suspension* when it requests a lock that is already held by another application process and cannot be shared. The suspended process temporarily stops running. A suspended process resumes when all processes that hold the conflicting lock release them or the requesting process experiences a timeout or deadlock and the process resumes and handles an error condition.

Timeout
An application process encounters a *timeout* when it terminates because of a suspension that exceeds a preset interval. Db2 terminates the process, issues messages, and returns error codes. Commit and rollback operations do not timeout. The STOP DATABASE command, however, can time out, in which case Db2 sends messages to the console. When this happens Db2 retries the STOP DATABASE command as many as 15 times.

Deadlock
A *deadlock* occurs when two or more application processes each hold locks on resources that the others need and without which they cannot proceed. After a preset time interval, Db2 can roll back the current unit of work for one of the processes or request a process to terminate. In determining which process to roll back or terminate, Db2 assesses many characteristics of the processes that are involved in the deadlock and chooses the one that, if terminated, will cause the least impact relative to the other processes. By choosing a process to roll back or terminate, Db2 frees the locks and allows the remaining processes to continue.

Although some locking problems can occur, you can avoid system and application locking problems.

The following scenarios illustrate the importance of locks.

Scenario: Avoidance of the loss of updated data

Two users, Kathy and Frank, are both trying to access the same Db2 table. Here is what happens:

1. Kathy reads the data value, 100, into a host variable.
2. Frank reads the same column value into a host variable.
3. Kathy adds 10 to the host variable value and saves the new value, 110, in the Db2 table column.
4. Frank adds 20 to the host variable value and saves the new value, 120, in the Db2 table column.

This scenario does not use locking. It shows that the updated value in the column depends on which user commits the data first. If Kathy commits first, the updated column value is 120, and Kathy's update is lost. If Frank commits first, the updated column value is 110, and Frank's update is lost.

The scenario changes if it includes locking. When you read the process below, assume the use of an updatable cursor. Here is what happens:

1. Kathy reads column value 100 into a host variable with the intention of updating the value. Db2 then grants an update lock to Kathy.
2. Frank wants to read the same column value into a host variable with the intention of updating the value. According to the compatibility matrix in the table above, Db2 does not grant Frank an update lock (U-lock) on the Db2 object that contains column value 100. Therefore, Frank must wait to read the column value until Kathy releases the lock.
3. Kathy adds 10 to the host variable value and wants to save the new value, 110, in the Db2 table column. At this point, Db2 changes the U-lock to an exclusive lock (X-lock) on the Db2 object that contains the column value.
4. Kathy commits the change. Db2 then releases the X-lock on the Db2 object that contains the column value. Next, Db2 grants the U-lock to Frank on the same object (unless Frank timed out while waiting for access). The host variable that Frank specified now contains the updated value of 110.
5. Frank adds 20 to the host variable value and wants to save the new value, 130, in the table column. Db2 changes the U-lock to an X-lock on the Db2 object that contains the column value.
6. Frank commits the change. Db2 then releases the X-lock on the Db2 object that contains the column value.

If this scenario did not include updatable cursors, Db2 would grant a share lock (S-lock) to Kathy instead of a U-lock in step 1. Db2 would also grant an S-lock to Frank in step 2. When both Kathy and Frank try to update the column value, they would encounter a deadlock. When a deadlock occurs, Db2 decides whether to roll back Kathy's work or Frank's work. A rollback occurs when Db2 reverses a change that an individual application process tried to make. If Db2 rolls back Kathy's changes, Kathy releases the locks, and Frank can then complete the process. Conversely, if Db2 rolls back Frank's changes, Frank releases the locks, and Kathy can complete the process.

Application programs can minimize the risk of deadlock situations by using the FOR UPDATE OF clause in the DECLARE CURSOR statement. The program does not actually acquire the U-lock until any other U-locks or X-locks on the data object are released.

Scenario: Avoidance of read access to uncommitted data

Two users, Kathy and Frank, are both trying to access the same Db2 table.

1. Kathy updates the value of 100 to 0 in the Db2 table column.
2. Frank reads the updated value of 0 and makes program decisions based on that value.
3. Kathy cancels the process and changes the value of 0 back to 100 for the Db2 table column.

This scenario does not include locks. It shows that Frank made an incorrect program decision. As a result, the business data in the database might be inaccurate.

When this scenario includes locking, this is what happens:

1. Kathy attempts to update the value of 100 to 0 in the table column. Db2 grants an X-lock to Kathy on the Db2 object that contains the column value that requires an update.

2. Frank tries to read the updated column value so that he can make program decisions based on that value. Db2 does not allow Frank to read the updated column value of 0. Frank tries to acquire an S-lock on the Db2 object that currently has the X-lock. Frank must wait until Kathy commits or rolls back the work.
3. Kathy cancels the process and changes the value of 0 back to the original value of 100 for the Db2 table column. Db2 makes the actual change to the data and releases the X-lock for Kathy. Db2 then grants the S-lock to Frank on the Db2 object that contains the column value. Frank then reads the value of 100.

When the scenario includes locks, Frank reads the correct data and can therefore make the correct program decision. As a result, the business data in the database is accurate.

Scenario: Avoidance of updates between multiple reads within a unit of work

In this scenario, Kathy wants to read the same data twice. No other program or user can change the data between the two reads.

GUPI

Assume that Kathy uses the following SQL statement:

```
SELECT * FROM EMP
  WHERE SALARY>
        (SELECT AVG(SALARY) FROM EMP);
```

GUPI

This SQL statement reads the EMP table twice:

1. It calculates the average of the values in the SALARY column of all rows in the table.
2. It finds all rows in the EMP table that have a value in the SALARY column that exceeds the average value.

If Kathy does not lock the data between the two read processes, another user can update the EMP table between the two read processes. This update can lead to an incorrect result for Kathy.

Kathy could use Db2 locks to ensure that no changes to the table occur in between the two read processes. Kathy can choose from these options:

- Using the package or plan isolation level of repeatable read (RR) or using the WITH RR clause in the SQL SELECT statement.
- GUPI
- Locking the table in share or exclusive mode, using one of these statements:
  - `LOCK TABLE EMP IN SHARE MODE`
  - `LOCK TABLE EMP IN EXCLUSIVE MODE`

GUPI

Related concepts

Concurrency and locks (Db2 Performance)
Lock modes and compatibility of locks (Db2 Performance)
Lock contention, suspension, timeout, and deadlock (Db2 Performance)

Related tasks

Programming for concurrency (Db2 Performance)
Designing databases for concurrency (Db2 Performance)

Improved performance through concurrency control

Concurrency control relies on the *isolation level* and *current data* options of applications to determined how much to isolate different applications that access the same data from each other. Too much isolation

might result in contention, whereas too little isolation might result in the return of non-current data to applications.

The *ISOLATION option* of an application specifies the degree to which operations are isolated from the possible effects of other operations that act concurrently. The ISOLATION options specified how soon Db2 can release S and U locks on rows or pages. Regardless of the isolation level that you specify, outstanding claims on Db2 objects can inhibit the execution of Db2 utilities or commands.

The *CURRENTDATA option* of an application specifies whether data currency is required for read-only and ambiguous cursors when the ISOLATION(CS) option is used. This option enables a trade-off between the improved ability of multiple applications to access the same data concurrently and the risk that non-current data might be returned to the application.

The basic recommendation is to bind most applications with the ISOLATION(CS) and CURRENTDATA(NO) options. ISOLATION(CS) typically enables Db2 to release acquired locks as soon as possible. The CURRENTDATA(NO) typically enables Db2 to acquire the fewest number of locks, for better *lock avoidance*.

Db2 provides the following isolation levels:

Cursor stability (CS)
The ISOLATION (CS) or *cursor stability* option allows maximum concurrency with data integrity. Under the ISOLATION (CS) option, a transaction holds locks only on its uncommitted changes and on the current row of each of its cursors.

Uncommitted read (UR)
The ISOLATION (UR) or *uncommitted read* option allows an application to read while acquiring few locks, at the risk of reading uncommitted data. UR isolation applies only to the following read-only operations: SELECT, SELECT INTO, or FETCH from a read-only result table.

Read stability (RS)
The ISOLATION (RS) or *read stability* option enables an application to read the same pages or rows more than once and prevents updates or deletes to qualifying rows by other processes. However, other applications can insert or update rows that did not satisfy the search condition of the original application.

Repeatable read (RR)
The ISOLATION (RR) or *repeatable read* option allows the application to read the same pages or rows more than once without allowing any update, insert, or delete operations by other processes. All accessed rows or pages are locked, even if they do not satisfy the predicate. Under the ISOLATION (RR) option, the data that an application references cannot be updated by any other applications before the application reaches a commit point.

Db2 uses and depends on locks because of the requirement for data integrity. However, locks are sometimes the cause of problems with contention, such as deadlocks, timeouts, and suspensions. To minimize these problems and promote concurrency, database designers and application designers can take various actions.

Related tasks

Improving concurrency (Db2 Performance)
Choosing an ISOLATION option (Db2 Performance)
Designing databases for concurrency (Db2 Performance)
Specifying the size of locks for a table space (Db2 Performance)
Programming for concurrency (Db2 Performance)

Related reference

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Concurrency recommendations for database designers

Database designers can take certain general actions to promote concurrency without compromising data integrity.

Procedure

To promote concurrency without compromising data integrity, use any of the following approaches:

- Keep like things together in the database.
- Keep unlike things apart from each other in the database.
- Use LOCKSIZE ANY or PAGE as a design default. Consider LOCKSIZE ROW only when applications encounter significant lock contention, including deadlock and timeout.
- Examine small tables, looking for opportunities to improve concurrency by reorganizing data or changing the locking approach.
- Partition secondary indexes to promote partition independence and reduce lock contention.
- Minimize update activity that moves rows across partitions.
- Store fewer rows of data in each data page.

Related tasks

Designing databases for concurrency (Db2 Performance)
Improving concurrency (Db2 Performance)

Concurrency recommendations for application designers

Application designers can take certain general actions to promote concurrency without compromising data integrity.

Procedure

To promote concurrency without compromising data integrity, use any of the following approaches:

- Program applications to access data in the same order.
- Commit work as soon as doing so is practical, to avoid unnecessary lock contention, even in read-only applications.
- Include logic in your application program to retry after a deadlock or timeout to attempt recovery from the contention situation without assistance.
- Bind most applications with the ISOLATION(CS) and CURRENTDATA(NO) options.

  These options enable Db2 to release locks early and avoid taking locks in many cases.
- Use global transactions, which enables Db2 and other transaction managers to participate in a single transaction and thereby share the same locks and access the same data.

Related tasks

Programming for concurrency (Db2 Performance)
Improving concurrency (Db2 Performance)

Ways to improve query performance

Access paths have a significant impact on Db2 performance. Db2 chooses access paths, but you can use tools to understand how access paths affect performance in certain situations.

An *access path* is the path that Db2 uses to locate data that is specified in SQL statements. An access path can be indexed or sequential.

Two important factors in the performance of an SQL statement are the amount of time that Db2 uses to determine the access path at run time and the efficiency of the access path. Db2 determines the access path for a statement either when you bind the plan or package that contains the SQL statement or when the SQL statement executes.

The time at which Db2 determines the access path depends on whether the statement is executed statically or dynamically and whether the statement contains input host variables.

The access path that Db2 chooses determines how long the SQL statement takes to run. For example, to execute an SQL query that joins two tables, Db2 has several options. Db2 might make any of the following choices to process those joins:

- Scan the PARTS table for every row that matches a row in the PRODUCTS table.
- Scan the PRODUCTS table for every row that matches a row in the PARTS table.
- Sort both tables in PROD# order; then merge the ordered tables to process the join.

Choosing the best access path for an SQL statement depends on a number of factors. Those factors include the content of any tables that the SQL statement queries and the indexes on those tables.

Db2 also uses extensive statistical information about the database and resource use to make the best access choices.

In addition, the physical organization of data in storage affects how efficiently Db2 can process a query.

Related tasks

Writing efficient SQL queries (Db2 Performance)
Maintaining data organization and statistics (Db2 Performance)
Programming applications for performance (Db2 Performance)

Tools for improving query performance

Several performance analysis tools can help you improve SQL performance.

IBM Data Studio or IBM Data Server Manager

IBM Data Studio and IBM Data Server Manager are sets of tools that help you to perform a wide range of database management activities, including query performance tuning.

Db2 Query Workload Tuner for z/OS

Db2 Query Workload Tuner for z/OS helps database administrators and SQL developers optimize the performance of SQL statements in applications that query Db2 for z/OS subsystems.

Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS

IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS is a performance analysis, monitoring, and tuning tool for Db2 for z/OS environments.

EXPLAIN statement and EXPLAIN tables

In Db2 for z/OS, EXPLAIN statements produces the following information:

- A plan, package, or SQL statement when it is bound. The output appears in a table that you create, called a *plan table*.
- The estimated cost of executing a SELECT, INSERT, UPDATE, or DELETE statement. The output appears in a table that you create, called a *statement table*.
- User-defined functions that are referred to in an SQL statement, including the specific function name and schema. The output appears in a table that you create, called a *function table*.

Db2 SQL Performance Analyzer for z/OS

Db2 SQL Performance Analyzer for z/OS provides you with an extensive analysis of SQL queries without executing them. This analysis aids you in tuning your queries to achieve maximum performance. Db2 SQL Performance Analyzer helps you reduce the escalating costs of database queries by estimating their cost before execution.

Using Db2 SQL Performance Analyzer for z/OS helps you:

- Estimate how long queries are likely to take
- Prevent queries from running too long
- Analyze new access paths
- Code queries efficiently using hints and tips that the tool provides

Related concepts

Investigating SQL performance by using EXPLAIN (Db2 Performance)

Interpreting data access by using EXPLAIN (Db2 Performance)

IBM Data Server Manager

IBM Data Server Manager is an integrated database management tools platform that is used for Db2 for z/OS subsystems. With Data Server Manager, you can monitor performance, track configuration changes, administer databases, and optimize the database environment.

IBM Data Studio

IBM Data Studio is a set of powerful set of tools that help you manage enterprise data, databases, and data-driven applications.

Related tasks

Investigating access path problems (Db2 Performance)

Related reference

Facilities and tools for Db2 performance monitoring (Db2 Performance)

EXPLAIN (Db2 SQL)

EXPLAIN tables (Db2 Performance)

Query and application performance analysis

Performance analysis for queries and applications begins by answering some basic questions.

To improve query performance and application performance, you need to answer some basic questions to determine how well your queries and applications perform.

Tip: You can use tools such as IBM Data Server Manager and related Db2 Tools for z/OS solutions to simplify this task.

- "IBM Data Server Manager" on page 305
- Db2 Tools for z/OS

Are the catalog statistics up-to-date?

Keeping object statistics current is an important activity. Db2 needs those statistics to choose an optimal access path to data.

Db2 utilities can collect statistics about Db2 objects. The RUNSTATS utility is designed for collecting statistics, but you can also collect inline statistics when you run certain other utilities, such as the LOAD, REORG INDEX, REORG TABLESPACE, and REBUILD INDEX utilities. These statistics are stored in the Db2 catalog. Db2 uses this information during the bind process to choose an access path. If you do not collect statistics and then rebind your packages or plans, Db2 does not have the information that it needs to choose the most efficient access path. Lack of statistical information can result in unnecessary I/O operations and excessive processor consumption.

Recommendation: Collect statistics at least once for each table and its associated indexes. How often you rerun the utility depends on how current you need the catalog data to be. If data characteristics of a table vary significantly over time, you must keep the catalog current with those changes. Collecting statistics is most beneficial for the following objects:

- Table spaces that contain frequently accessed tables
- Tables that are involved in sort operations
- Tables with many rows
- Tables that are queried by SELECT statements that include many search arguments

- Tables with indexes

Related information:

Investigating access path problems (Db2 Performance)
Maintaining Db2 database statistics (Db2 Performance)
RUNSTATS (Db2 Utilities)

Does the estimated filter factor match the filtering at run time?

Db2 uses estimated filter factors to choose efficient access paths. When the estimated and actual filtering differs, Db2 might choose an access path that performs poorly because the cost estimates are inaccurate. When the estimated and actual factors differ, you might collect more statistics or reoptimize statements at run time to improve access path selection.

Related information:

Predicate filter factors (Db2 Performance)
Investigating access path problems (Db2 Performance)
Reoptimizing SQL statements at run time (Db2 Performance)
Overriding predicate selectivities at the statement level (Db2 Performance)

Is the query coded as simply and efficiently as possible?

Ensure that the SQL query is coded as simply and efficiently as possible. Make sure that:

- Unused columns are not selected.
- No unneeded ORDER BY or GROUP BY clauses are in the query.
- No unneeded predicates are in the query.

Related information:

Writing efficient SQL queries (Db2 Performance)
Coding SQL statements to avoid unnecessary processing (Db2 Performance)
Using predicates efficiently (Db2 Performance)

Are you using materialized query tables?

Define materialized query tables to improve the performance of dynamic queries that operate on large amounts of data and that involve multiple joins. Db2 generates the results of all or parts of the queries in advance and stores the results in materialized query tables. Db2 determines when using precomputed results is likely to optimize the performance of dynamic queries.

Related information:

Using materialized query tables to improve SQL performance (Db2 Performance)

Is access through an index?

An index provides efficient access to data. Db2 uses different types of index scans, each of which affects performance differently. Sometimes Db2 can avoid a sort by using an index.

If a query is satisfied by using only the index, Db2 uses a method called *index-only access*.

- For a SELECT operation, if all the columns that are needed for the query can be found in the index, Db2 does not need to access the table.
- For an UPDATE or DELETE operation, an index-only scan can be performed to search for qualifying rows to update or delete. After the qualifying rows are identified, Db2 must retrieve those rows from the table space before they are updated or deleted.

Other types of index scans that Db2 might use are matching or nonmatching index scans.

- In a matching index scan, the query uses predicates that match the index columns. Predicates provide filtering; Db2 needs to access only specific index and data pages.
- In a nonmatching index scan, Db2 reads all index keys and their rows of data. This type of scan is less likely to provide an efficient access path than a matching index scan.

In addition to providing selective access to data, indexes can also order data, and sometimes they eliminate the need for sorting. You can avoid some sorts if index keys are in the order that is needed by ORDER BY, GROUP BY, a join operation, or DISTINCT in an aggregate function. When you want to prevent a sort, consider creating an index on the columns that are necessary to provide that ordering.

Related information:

Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX') (Db2 Performance)
Indexes on table columns (Db2 Administration Guide)

Is a table space scan used?

When index access is not possible, Db2 uses a table space scan. Db2 typically uses the sequential prefetch method to scan table spaces.

GUPI

Example: Assume that table T has no index on column C1. Db2 uses a table space scan in the following example:

```
SELECT * FROM T WHERE C1 = VALUE;
```

In this case, every row in table T must be examined to determine if the value of C1 matches the given value.

GUPI

A table space scan on a partitioned table space can be more efficient than a scan on a nonpartitioned table space. Db2 can take advantage of the partitions by limiting the scan of data in a partitioned table space to one or more partitions.

Related information:

Table space scan access (ACCESSTYPE='R' and PREFETCH='S') (Db2 Performance)
Sequential prefetch (PREFETCH='S') (Db2 Performance)

Are sorts performed?

Minimizing the need for Db2 to use sorts to process a query can result in better performance. In general, try to create indexes that match the predicates in your queries before trying to avoid sorts in your queries.

Related information:

Sorts of data (Db2 Performance)
Sort access (Db2 Performance)

Is data accessed or processed in parallel?

Parallel processing applies to read-only queries. Db2 can use parallel I/O and CPU operations to improve performance. For example, Db2 can use multiple parallel operations to access data from a table or index. The response time for data-intensive or processor-intensive queries can be reduced.

Related information:

Programming for parallel processing (Db2 Performance)

Are host variables used?

When you specify the bind option REOPT(VARS), Db2 determines the access paths at both bind time and run time for statements that contain one or more host variables, parameter markers, or special registers. At run time, Db2 uses the values in those variables to determine access paths.

Db2 spends extra time determining the access path for statements at run time. But if Db2 finds a better access path using the variable values, you might see an overall performance improvement.

For static SQL applications with host variables, if you specify REOPT(VARS), Db2 determines the access path at bind time and again at run time, using the values of input variables.

For static SQL applications with no host variables, Db2 determines the access path when you bind the plan or package. This situation yields the best performance because the access path is already determined when the program runs.

For applications that contain dynamic SQL statements with host variables, using REOPT(VARS) is the recommended approach for binding.

Related information:

Reoptimizing SQL statements at run time (Db2 Performance)

Are dynamic SQL statements used?

For dynamic SQL statements, Db2 determines the access path at run time, when the statement is prepared.

When an application performs a commit operation, it must issue another PREPARE statement if that SQL statement is to be executed again. For a SELECT statement, the ability to declare a cursor WITH HOLD provides some relief but requires that the cursor be open at the commit point. Using the WITH HOLD option also causes some locks to be held for any objects that the prepared statement depends on. Also, the WITH HOLD option offers no relief for SQL statements that are not SELECT statements.

You can use the dynamic statement cache to decrease the number of times that those dynamic statements must be prepared. Using the dynamic statement cache is useful when you execute the same SQL statement often.

Db2 can save prepared dynamic statements in a cache. The cache is a Db2-wide cache that all application processes can use to store and retrieve prepared dynamic statements. After an SQL statement is prepared and is automatically stored in the cache, subsequent prepare requests for that same SQL statement can use the statement in the cache to avoid the costly preparation process. Different threads, plans, or packages can share cached statements.

The SELECT, UPDATE, INSERT, and DELETE statements are eligible for caching.

Related information:

Held and non-held cursors (Db2 Application programming and SQL)

Related concepts

Interpreting data access by using EXPLAIN (Db2 Performance)

Related tasks

Programming applications for performance (Db2 Performance)

Investigating access path problems (Db2 Performance)

Related reference

EXPLAIN (Db2 SQL)

Using EXPLAIN to understand the access path

You can use the EXPLAIN statement to determine the access paths for the SELECT parts of your statements.

Tip: You can use tools such as IBM Data Server Manager and related Db2 Tools for z/OS solutions to simplify this task.

- "IBM Data Server Manager" on page 305
- Db2 Tools for z/OS

This information describes what EXPLAIN provides and how you can obtain information from EXPLAIN. The information in the plan table can help you when you need to perform the following tasks:

- Determine the access path that Db2 chooses for a query
- Design databases, indexes, and application programs
- Determine when to rebind an application

For each access to a single table, EXPLAIN indicates whether Db2 uses index access or a table space scan. For indexes, EXPLAIN indicates how many indexes and index columns are used and what I/O methods are used to read the pages. For joins of tables, EXPLAIN indicates the join method and type, the order in which Db2 joins the tables, and the occasions when and reasons why it sorts any rows.

The following steps summarize how to obtain information from EXPLAIN:

1. Create the plan table.

   Before you can use EXPLAIN, you must create a plan table to hold the results of EXPLAIN.
2. Populate the plan table.

   You can populate the plan table by executing the SQL statement EXPLAIN. You can also populate a plan table when you bind or rebind a plan or package by specifying the option EXPLAIN(YES). EXPLAIN obtains information about the access paths for all explainable SQL statements in a package or in the DBRMs of a plan.
3. Select information from the plan table.

   Several processes can insert rows into the same plan table. To understand access paths, you must retrieve the rows for a particular query in an appropriate order.

EXPLAIN helps you answer questions about query performance; the answers give you the information that you need to make performance improvements. EXPLAIN indicates whether Db2 used an index to access data, whether sorts were performed, whether parallel processing was used, and so on.

As you gain experience working with Db2, you can use the plan table to give optimization hints to Db2 that influence access path selection.

Note: Although EXPLAIN is useful for obtaining access path information, it requires a great amount of background knowledge about Db2. However, analysis tools are available, such as the access plan graph feature of IBM Data Studio or IBM Data Server Manager, that can assist you with access path analysis.

Related concepts

Investigating SQL performance by using EXPLAIN (Db2 Performance)

Interpreting data access by using EXPLAIN (Db2 Performance)

Related tasks

Investigating access path problems (Db2 Performance)

Related reference

EXPLAIN tables (Db2 Performance)

EXPLAIN (Db2 SQL)

EXPLAIN bind option (Db2 Commands)

IBM Data Studio product overview (IBM Data Studio)

Related information

IBM Data Server Manager Documentation

Questions for investigating data access

You can focus your investigation of access paths that Db2 uses to process SQL statements by using the PLAN_TABLE data to answer certain questions.

Tip: You can use tools such as IBM Data Server Manager and related Db2 Tools for z/OS solutions to simplify this task.

- "IBM Data Server Manager" on page 305
- Db2 Tools for z/OS

Tip: You can use the *visual explain* feature of tools such as IBM Data Server Manager to create visual diagrams of the access paths for your SQL statements.

PSPI

However, if you do analyze PLAN_TABLE data directly, you can use the following questions to guide your initial analysis of the access paths:

How are indexes used to access the data?

The ACCESSTYPE and MATCHOLS values contain information about the use of indexes in an access path.

1. Is an index used? For information about interpreting index access, see Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX') (Db2 Performance).
2. How many indexes are used? For information about interpreting access through multiple indexes, see Multiple index access (ACCESSTYPE='M', 'MX', 'MI', 'MU', 'DX', 'DI', or 'DU') (Db2 Performance).
3. How many index columns are used in matching? For more information about finding the number of matching index columns, see Matching index scan (MATCHCOLS>0) (Db2 Performance) and The number of index columns used for matching (MATCHCOLS=n) (Db2 Performance).
4. Is the query satisfied by the index alone? For more information about analyzing index-only access, see Index-only access (INDEXONLY='Y') (Db2 Performance).
5. How many index screening columns are used? For more information about analyzing index screening, see Index screening (Db2 Performance).

Is direct row access used?

Direct row access can only be used only when the table contains a column of the ROWID data type. For information about direct row access, see Direct row access (PRIMARY_ACCESSTYPE='D') (Db2 Performance) and "ROWID data type" on page 25.

What possibly costly operations are used?

1. Is a view or nested table expression materialized? For more information about analyzing materialization, see View and nested table expression access (Db2 Performance).
2. Was a scan limited to certain partitions? For information about analyzing the use of page-range screening, see Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U') (Db2 Performance).
3. What prefetch type is expected? For information about analyzing the use of prefetch, see Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U') (Db2 Performance).
4. Is data accessed or processed in parallel? For information about analyzing the use of parallelism in the access path, see Parallel processing access (PARALLELISM_MODE='I', or 'C') (Db2 Performance).
5. Is data sorted? For information about analyzing the use of sort operations, see Sort access (Db2 Performance).
6. Is a subquery transformed to a join? For information about analyzing subquery access, see Subquery access (Db2 Performance).
7. When are aggregate functions evaluated? For information about analyzing when Db2 evaluates aggregate functions, see Aggregate function access (COLUMN_FN_EVAL) (Db2 Performance).
8. Is a complex trigger WHEN clause used? For information about identifying the use of triggers in a WHEN clause, see Complex trigger WHEN clause access (QBLOCKTYPE='TRIGGR') (Db2 Performance).

What are the object dependencies for the access path?

Access paths have dependencies on the objects that are identified in the TNAME and ACCESSNAME columns of the PLAN_TABLE. However, access paths might also depend on objects that Db2 does not actually use when it processes the selected access paths. Such dependencies are not be shown in EXPLAIN output, but they are recorded in the SYSPACKDEP catalog table (Db2 SQL).

PSPI

Related concepts

Interpreting data access by using EXPLAIN (Db2 Performance)

Related reference

PLAN_TABLE (Db2 Performance)

EXPLAIN (Db2 SQL)

Hash access paths

Db2 can use hash access paths to directly access a single row in a table and avoid scanning the table or index.

Db2 can generate a hash access path from the equal predicate of an SQL statement that fetches only a single row of a table. This access method is faster and more efficient than scanning the table or the index.

Hash access can reduce the query access time and the CPU load for queries that require access to a single row. However, tables that are enabled for hash access might require as much as twice the amount of disk space as tables that are not organized for hash access.

You can enable hash access on tables when you create new table spaces. You can also alter existing tables to enable hash access. However, hash access is available only on universal table spaces that are partitioned by growth or partitioned by range. You can enable hash access on tables that have indexes, but Db2 does not allow hash access on tables on which clustering is enabled.

Related concepts

Hash access (ACCESSTYPE='H', 'HN', or 'MH') (Db2 Performance)

Db2 hash spaces (deprecated)

A *hash space* is a defined disk space that organizes table data for hash access. Hash-organized table spaces are deprecated and likely to be unsupported in the future.

Related tasks

Organizing tables for hash access to individual rows (deprecated) (Db2 Performance)

Creating tables that use hash organization (deprecated) (Db2 Administration Guide)

Altering tables for hash access (deprecated) (Db2 Administration Guide)

Monitoring hash access (deprecated) (Db2 Performance)

Managing space and page size for hash-organized tables (Db2 Performance)

Related information

DB2 10 for z/OS Performance Topics (IBM Redbooks)

DB2 10 for z/OS Technical Overview (IBM Redbooks)

Chapter 5. Programming for Db2 for z/OS

You use application programs that contain SQL statements to perform data definition or data manipulation operations on Db2 for z/OS objects.

Where can I find application programming samples?
See Db2 programming samples (Db2 Programming samples).

What application development tools can I use?

See "Tools and IDEs for developing Db2 applications" on page 233.

What do I need to do in an SQL application?

In an application, you need to complete these tasks:

1. Declare variables for passing data to and from the Db2 database system.
2. Execute SQL statements.
3. Check for and handle SQL errors

What programming languages can I use to write SQL applications?

You can write application programs for Db2 for z/OS data servers in any of the following programming languages:

- APL2®
- Assembler
- C and C++
- C# "2" on page 229 (through .NET)
- COBOL
- Fortran
- High-level Assembler (or HLASM)
- Java "1" on page 229
- Perl "2" on page 229
- PHP "2" on page 229
- PL/I
- Python "2" on page 229
- REXX
- Ruby on Rails "2" on page 229
- SQL procedural language
- Visual Basic "2" on page 229

1. By connecting to Db2 for z/OS using IBM Data Server Driver for JDBC and SQLJ.
2. By remote connections to Db2 for z/OS using Db2 for Linux, UNIX, and Windows client drivers.

How do I pass data between SQL applications and a Db2 table?

You use host variables and indicator variables to pass data between SQL applications and Db2 tables. *Host variables* are application variables that contain the data that is exchanged with the Db2 table. *Indicator variables* are application variables whose values denote whether there is data in the host variables.

© Copyright IBM Corp. 2001, 2020

What are static and dynamic SQL applications?

In a static SQL application, the SQL statements are written directly in the application. When the application runs, the user can change only the variable values. In a dynamic SQL application, the user provides the SQL statements when the application runs. In general, dynamic SQL applications are more flexible than static SQL applications. However, dynamic applications are harder to write, and often run more slowly because the application has more processing to do at run time.

The following table lists some sample programs that are shipped with Db2, which demonstrate how to write static and dynamic SQL applications. You can find the source code in data set *prefix*.SDSNSAMP.

Program name	Language	Type of SQL
DSN8BC3	COBOL	Static
DSN8BD3	C	Static
DSN8BP3	PL/I	Static
DSNTEP2	PL/I	Dynamic

How do I prepare an SQL application for execution?

Preparing most SQL applications involves the following tasks:

1. For each source file in the application:
   - Modify a copy of the source file to convert the SQL statements into host language calls to an attachment facility.
   - Create a database request module (DBRM).

   The preferred way to accomplish this step is to direct the compiler to call the Db2 coprocessor. However, you can include a separate step that runs the Db2 precompiler before the step that runs the compiler.
2. Compile the modified source files to produce z/OS object code.
3. Link edit the object code for all the object modules to produce a z/OS load module.
4. Bind the DBRM for each source file into a Db2 package, and binding the Db2 packages into a Db2 plan.

Note:

For C or C++ programs that use ODBC, Java programs that use JDBC or SQLJ, and REXX programs, the steps for preparing an SQL application for execution are different. The following table lists those steps.

Type of application	Program preparation steps
ODBC	Compile, prelink, link edit
JDBC	Compile
SQLJ	Translate, compile, customize, bind
pureQuery	Use IBM Data Studio
REXX	No program preparation

Where do I find more information?

The following tables provide more information about application programming for each of the commonly used languages.

Table 26. Resources for writing Db2 applications in C

Subject	See the following topics
Declaring variables	Declaring host variables and indicator variables in C (Db2 Application programming and SQL), Equivalent SQL and C data types (Db2 Application programming and SQL)
Executing SQL statements	Programming C and C++ applications that issue SQL statements (Db2 Application programming and SQL)
Checking for and handling SQL errors	Defining the SQL communications area, SQLSTATE, and SQLCODE in C (Db2 Application programming and SQL), Checking the execution of SQL statements (Db2 Application programming and SQL)
Preparing an application to run	Preparing an application to run on Db2 for z/OS (Db2 Application programming and SQL)

Table 27. Resources for writing ODBC applications

Subject	See the following topics
Declaring variables	Data types and data conversion (Db2 Programming for ODBC)
Executing SQL statements	Transaction processing in Db2 ODBC (Db2 Programming for ODBC)
Checking for and handling SQL errors	Diagnostics (Db2 Programming for ODBC)
Preparing an application to run	Overview of preparing and executing a Db2 ODBC application (Db2 Programming for ODBC)

Table 28. Resources for writing Db2 applications in COBOL

Subject	See the following topics
Declaring variables	Declaring host variables and indicator variables in COBOL (Db2 Application programming and SQL), Equivalent SQL and COBOL data types (Db2 Application programming and SQL)
Executing SQL statements	Programming COBOL applications that issue SQL statements (Db2 Application programming and SQL)
Checking for and handling SQL errors	Defining the SQL communications area, SQLSTATE, and SQLCODE in COBOL (Db2 Application programming and SQL), Checking the execution of SQL statements (Db2 Application programming and SQL)
Preparing an application to run	Preparing an application to run on Db2 for z/OS (Db2 Application programming and SQL)

Table 29. Resources for writing JDBC applications

Subject	See the following topics
Starting point	JDBC application programming (Db2 Application Programming for Java)
Declaring variables	Variables in JDBC applications (Db2 Application Programming for Java), Data types that map to database data types in Java applications (Db2 Application Programming for Java)
Executing SQL statements	JDBC interfaces for executing SQL (Db2 Application Programming for Java)
Checking for and handling SQL errors	Exceptions and warnings under the IBM Data Server Driver for JDBC and SQLJ (Db2 Application Programming for Java)
Preparing an application to run	Program preparation for JDBC programs (Db2 Application Programming for Java)

Table 30. Resources for writing SQLJ applications

Subject	See the following topics
Starting point	SQLJ application programming (Db2 Application Programming for Java)
Declaring variables	Variables in SQLJ applications (Db2 Application Programming for Java), Data types that map to database data types in Java applications (Db2 Application Programming for Java)
Executing SQL statements	SQL statement execution in SQLJ applications (Db2 Application Programming for Java)
Checking for and handling SQL errors	Handling SQL errors and warnings in SQLJ applications (Db2 Application Programming for Java)
Preparing an application to run	Program preparation for SQLJ programs (Db2 Application Programming for Java)

Table 31. Resources for programming with JSON

Subject	See the following topics
Starting point	JSON application development for IBM data servers (Programming with JSON)
JSON installation requirements	JSON installation requirements (Programming with JSON)
Working with JSON documents	Working with JSON documents (Programming with JSON)

Table 32. Resources for writing pureQuery applications

Subject	See the following topics
Starting point	Using IBM Data Studio to develop Java applications that access data

Table 33. Resources for writing Db2 applications in PL/I

Subject	See the following topics
Declaring variables	Declaring host variables and indicator variables in PL/I (Db2 Application programming and SQL), Equivalent SQL and PL/I data types (Db2 Application programming and SQL)
Executing SQL statements	Programming PL/I applications that issue SQL statements (Db2 Application programming and SQL)
Checking for and handling SQL errors	Defining the SQL communications area, SQLSTATE, and SQLCODE in PL/I (Db2 Application programming and SQL), Checking the execution of SQL statements (Db2 Application programming and SQL)
Preparing an application to run	Preparing an application to run on Db2 for z/OS (Db2 Application programming and SQL)

Table 34. Resources for writing Db2 applications in REXX

Subject	See the following topics
Declaring variables	Passing the data type of an input data type to Db2 for REXX programs (Db2 Application programming and SQL), Equivalent SQL and REXX data types (Db2 Application programming and SQL)
Executing SQL statements	Programming REXX applications that issue SQL statements (Db2 Application programming and SQL)
Checking for and handling SQL errors	Defining the SQL communications area, SQLSTATE, and SQLCODE in REXX (Db2 Application programming and SQL), Checking the execution of SQL statements (Db2 Application programming and SQL)
Preparing an application to run	No program preparation is necessary.

Other learning resources

For information on training in Db2 application programming, see Db2 for z/OS training paths.

Tools and IDEs for developing Db2 applications

Application developers can use various application development tools and integrated development environments (IDEs) to create database applications, and stored procedures that access and use data in Db2 for z/OS.

The following application development tools and IDEs support the development of Db2 for z/OS database applications and routines.

IBM Data Studio

IBM Data Studio is a suite of Eclipse-based tools that are for development database administrators and application developers. You can use IBM Data Studio for the following tasks:

- Developing pureQuery applications in a Java project

- Creating, testing, debugging, and deploying routines, such as stored procedures and user-defined functions
- Creating, editing, and running SQL queries
- Connecting to data sources and browsing data objects and their properties
- Creating and altering data objects

Rational Developer for System z

Rational® Developer for System z® can improve efficiency and helps with mainframe development, web development, and integrated mixed workload or composite development. By using Rational Developer for System z, you can accelerate the development of your web applications, traditional COBOL and PL/I applications, web services, and XML-based interfaces.

Rational Developer for System z provides a common workbench and an integrated set of tools that support end-to-end, model-based application development, run time testing, and rapid deployment of On Demand applications. With the interactive, workstation-based environment, you can quickly access your z/OS data.

Rational Application Developer for WebSphere Software

IBM Rational software provides a full range of tools to meet your analysis, design, and construction needs, whether you are an application developer, application architect, systems engineer, or database designer. IBM Rational Application Developer for WebSphere Software helps developers to quickly design, develop, analyze, test, profile, and deploy high-quality web, Service-oriented Architecture (SOA), Java, J2EE, and portal applications.

By using Rational Application Developer, you can increase productivity, minimize your learning curve, and shorten development and test cycles so that you can deploy applications quickly.

WebSphere Studio Application Developer

WebSphere Studio Application Developer is a fully integrated Java development environment. Using WebSphere Studio Application Developer, you can build, compile, and test J2EE (Java 2 Enterprise Edition) applications for enterprise On Demand business applications with:

- JSP (JavaServer Pages) files
- EJB (Enterprise JavaBeans) components
- 100% Pure Java applets and servlets

For more information see, IBM WebSphere Application Server Developer Tools for Eclipse.

Db2 Development add-in for Visual Studio .NET

You can use the Db2 Development add-in for Microsoft Visual Studio .NET to enhance integration with the Microsoft Visual Studio .NET development environment.

The add-in features make it easy for application programmers to work with Db2 servers and to develop Db2 routines and objects.

The key add-in features enable developers to perform the following tasks:

Building Db2 server-side objects

Db2 Connect provides a Db2 .NET Data Provider, which enables .NET applications to access Db2 for z/OS and workstation (Windows, UNIX, and Linux) operating systems.

Using the Solution Explorer, developers can use script files for building objects that include routines, triggers, tables, and views.

Accessing and manage Db2 data connections

The IBM Explorer provides access to IBM database connections and enables developers to perform the following tasks:

- Work with multiple Db2 connections
- View object properties
- Retrieve and update data from tables and views
- View source code for Db2 procedures and functions
- Generate ADO .NET code using a drag-and-drop technique

  For information about using ADO .NET for applications that connect to Db2 for z/OS, see the Db2 for Linux, UNIX, and Windows information in IBM Knowledge Center .

Launching Db2 development and administration tools
These tools include IBM Data Studio, IBM Data Server Manager, Replication Center, Command Center, Task Center, and Journal.

ODBC based development tools

A wide variety of tools are available for performing tasks such as querying a database. These tools include ODBC-based tools such as Lotus Approach, Microsoft Access, Microsoft Visual Basic, Microsoft Excel, and many others. The ODBC-based tools provide a simpler alternative to developing applications than using a high-level programming language. QMF for Workstation provides access to Db2 data for these tools. With all of these tools, you can specify Db2 for z/OS as the database to access.

Related concepts
Use of development tools to create a stored procedure
Workstation-based development tools can help you create, install, and test stored procedures for Db2 for z/OS.

Programming languages and methods for developing application programs that access Db2 for z/OS

You can use a wide variety of programming languages and techniques to develop application programs for Db2 for z/OS. In addition, several methods are available for communicating with Db2.

You can write application programs for Db2 for z/OS data servers in any of the following programming languages:

- APL2
- Assembler
- C and C++
- C# "2" on page 235 (through .NET)
- COBOL
- Fortran
- High-level Assembler (or HLASM)
- Java "1" on page 235
- Perl "2" on page 235
- PHP "2" on page 235
- PL/I
- Python "2" on page 235
- REXX
- Ruby on Rails "2" on page 235
- SQL procedural language
- Visual Basic "2" on page 235

1. By connecting to Db2 for z/OS using IBM Data Server Driver for JDBC and SQLJ.
2. By remote connections to Db2 for z/OS using Db2 for Linux, UNIX, and Windows client drivers.

You can use any of the following programming methods:

Static SQL
The source form of a static SQL statement is embedded within an application program that is written in a traditional programming language. (Traditional programming languages include C, C++, COBOL, Fortran, PL/I, and Assembler.) Static SQL is a good choice when you know what statements an application needs to execute before the application runs.

Dynamic SQL
Unlike static SQL, dynamic statements are constructed and prepared at run time. Dynamic SQL is a good choice when you do not know the format of an SQL statement when you write a program. It is also a good choice when the program needs to generate part or all of an SQL statement based on input from its users.

ODBC
ODBC is an application programming interface (API) that C and C++ application programs can use to access relational databases. ODBC is well suited to the client/server environment.

pureQuery
pureQuery is a high-performance data access platform for Java applications that makes it easier to develop, optimize, secure, and manage data access.

SQLJ and JDBC
Like ODBC and C++, the SQLJ and JDBC Java interfaces let you write portable application programs that are independent of any one database product.

- SQLJ application support lets you write static SQL applications in the Java programming language. With SQLJ, you can embed SQL statements in your Java applications.
- JDBC application support lets you write dynamic SQL applications in the Java programming language. JDBC is similar to ODBC, but it is designed specifically for use with Java.

Related concepts

Use of an application program as a stored procedure
A stored procedure is a compiled program that can execute SQL statements.

Dynamic SQL applications
With dynamic SQL, Db2 prepares and executes the SQL statements within a program while the program is running. Dynamic SQL is a good choice when you do not know the format of an SQL statement before you write or run a program.

Differences between static and dynamic SQL (Db2 Application programming and SQL)

Use of Java to execute static and dynamic SQL
Db2 for z/OS supports SQLJ and JDBC. In general, Java applications use SQLJ for static SQL, JDBC for dynamic SQL, and PureQuery for both static and dynamic SQL.

Use of ODBC to execute dynamic SQL
Open Database Connectivity (ODBC) lets you access data through ODBC function calls in your application. The ODBC interface eliminates the need for precompiling and binding your application and increases the portability of your application.

Introduction to Db2 ODBC (Db2 Programming for ODBC)

Static SQL applications
For most Db2 users, static SQL provides a straightforward, efficient path to Db2 data.

Related tasks

Planning for and designing Db2 applications (Db2 Application programming and SQL)

Performance information for SQL application programming

Efficient applications are an important first step to good system and application performance. As you code applications that access data in Db2, consider performance objectives in your application design.

The following topics can help you understand how application programmers can consider performance as they write applications that access data in Db2 for z/OS.

Concurrency and programming

The goal is to program and prepare applications in a way that:

- Protects the integrity of the data that is being read or updated from being changed by other applications.
- Minimizes the length of time that other access to the data is prevented.

For more information about data concurrency in Db2 and recommendations for improving concurrency in your application programs, see the following topics:

- Designing databases for concurrency (Db2 Performance)
- Concurrency and locks (Db2 Performance)
- Improving concurrency (Db2 Performance)
- Improving concurrency in data sharing environments (Db2 Data Sharing Planning and Administration)

Writing efficient queries

The predicates, subqueries, and other structures in SQL statements affect the access paths that Db2 uses to access the data.

For information about how to write SQL statements that access data efficiently, see the following topics:

- “Ways to improve query performance” on page 220
- Writing efficient SQL queries (Db2 Performance)

Analyzing access paths

By analyzing the access path that Db2 uses to access the data for an SQL statement, you can discover potential problems. You can use this information to modify your statement to perform better.

Tip: You can use tools such as IBM Data Server Manager and related Db2 Tools for z/OS solutions to simplify this task.

- “IBM Data Server Manager” on page 305
- Db2 Tools for z/OS

For information about how you can use EXPLAIN tables to analyze the access paths for your SQL statements, see the following topics:

- Investigating access path problems (Db2 Performance)
- 00C200A4 (Db2 Codes)
- Investigating SQL performance by using EXPLAIN (Db2 Performance)
- Interpreting data access by using EXPLAIN (Db2 Performance)
- EXPLAIN tables (Db2 Performance)
- EXPLAIN (Db2 SQL)

Distributed data access performance

The goal is to reduce the amount of network traffic that is required to access the distributed data, and to manage the use of system resources such as distributed database access threads and connections.

For information about improving the performance of applications that access distributed data, see the following topics:

- “Ways to reduce network traffic” on page 153
- Managing Db2 threads (Db2 Performance)
- Improving performance for applications that access distributed data (Db2 Performance)
- Improving performance for SQL statements in distributed applications (Db2 Performance)

Stored procedures performance

For information about stored procedures and Db2 performance, see the following topics:

- Implementing Db2 stored procedures ()
- Improving the performance of stored procedures and user-defined functions (Db2 Performance)

Related concepts

SQL: The language of Db2 (Db2 SQL)

Programming for Db2 for z/OS

You use application programs that contain SQL statements to perform data definition or data manipulation operations on Db2 for z/OS objects.

Related tasks

Programming applications for performance (Db2 Performance)

Planning for and designing Db2 applications (Db2 Application programming and SQL)

Preparation process for an application program

How you prepare an application program to run depends on the type of application. The program preparation steps for applications vary based on the type of programming language that is used.

Db2 applications require different methods of program preparation, depending on the type of the application.

Applications that contain embedded static or dynamic SQL statements
Db2 applications embed SQL statements in traditional language programs. To use these programs, you must follow the typical preparation steps (compile, link-edit, and run) plus the Db2 precompile and bind steps. Some languages can be precompiled and compiled in a single step by a coprocessor.

Applications in interpreted languages, such as REXX and APL2
REXX procedures use dynamic SQL. You do not precompile, compile, link-edit, or bind Db2 REXX procedures before you run them.

Applications that contain ODBC calls
These applications pass dynamic SQL statements as arguments. You do not precompile or bind ODBC applications. ODBC applications use a standard set of functions to execute SQL statements and related services at run time.

Java applications, which can contain JDBC calls or embedded SQL statements
Preparing a Java program that contains only JDBC methods is the same as preparing any other Java program. You can prepare the program using the `javac` command. JDBC applications do not require the precompile or bind steps.

You can use IBM pureQuery can create and prepare Java applications. IBM pureQuery enables SQL access to databases or Java objects that are in memory and facilitates SQL best practices.

Preparing an SQLJ program requires a precompile step and a bind step.

The following program preparations steps are required by host programming languages:

1. *Precompile* the program. Before you compile or assemble C, COBOL, Fortran, PL/I, and Assembler language programs, you must prepare the SQL statements that are embedded in the program. The Db2 precompiler prepares the SQL statements for the because most compilers do not recognize SQL statements, you must use the Db2 precompiler before you compile the program to prevent compiler errors. The precompiler scans the program and returns modified source code, which you can then compile and link-edit.

   As an alternative, you can use a host language Db2 coprocessor for C, C++, COBOL, and PL/I as you compile your program. The Db2 coprocessor performs Db2 precompiler functions at compile time.

   The main output from the precompiler is a *database request module (DBRM)*. A DBRM is a data set that contains SQL statements and host variable information that is extracted from the source program during program preparation. The purpose of a DBRM is to communicate your SQL requests to Db2 during the bind process.

2. *Bind* the application program. Before your Db2 application can run, you must bind the DBRM to a package. You might decide to put certain SQL statements together in the same program to precompile them into the same DBRM and then bind them into a single package. When the program runs, Db2 uses a timestamp to verify that the program matches the correct plan or package.

   A *collection* is a group of associated packages. By binding packages into package collections you can add packages to an existing application plan without needing to bind the entire plan again. If you include a collection name in the package list when you bind a plan, any package that is in the collection becomes available to the plan. The collection can even be empty when you first bind the plan. Later, you can add packages to the collection and drop or replace existing packages without binding the plan again.

   The CURRENT PACKAGE PATH special register specifies a value that identifies a list of collections that Db2 uses when resolving references to packages that you use to run SQL statements.

   For more information about the bind process, see Binding application packages and plans (Db2 Application programming and SQL).
3. *Compile* and *link-edit* the program. To enable your application to interface with the Db2 subsystem, you must use a link-edit procedure to build an executable load module that satisfies the requirements of your environment (such as CICS, IMS, TSO, or batch). The *load module* is a program unit that is loaded into main storage for execution.
4. *Run* the program: After you complete the preceding steps, you can run your Db2 application. You can use the followng methods to prepare an application to run:
   - Use Db2 Interactive (DB2I) panels, which lead you step by step from preparing the program to running the program.
   - Submit an application in the TSO foreground or in batch in the TSO background.
   - Start the program preparation command list (CLIST) in TSO foreground or batch.
   - Use the DSN command processor.
   - Use JCL procedures that you include in your data sets (such as SYS1.PROCLIB) at Db2 installation time.

You can also precompile and prepare an application program by using a Db2-supplied procedure. Db2 has a unique procedure for each supported language.

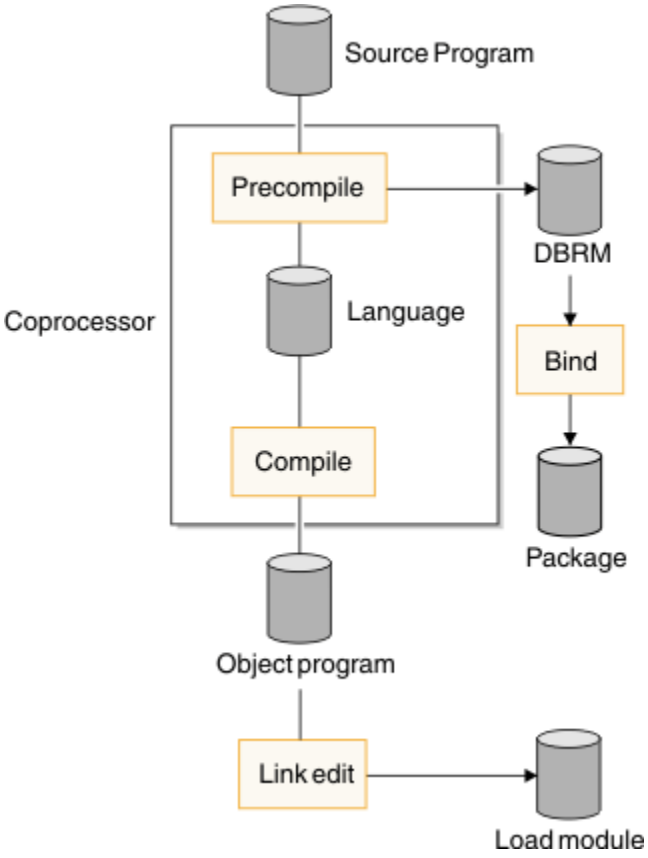


Figure 50. Overview of the program preparation process for applications that contain embedded SQL. The Db2 coprocessor can combine the precompile and compile steps for certain languages.

Db2 Bind Manager tool

The Db2 Bind Manager tool helps application programmers:

- Predict whether a bind of a DBRM results in a changed access path
- Run access path checks on a batch of DBRMs
- Eliminate unnecessary bind steps between application programs and the database
- Compare DBRMs to subsystems and load modules

Db2 Path Checker tool

The Db2 Path Checker helps you increase the stability of your Db2 environments and avoid painful and costly disruptions. The Db2 Path Checker can help you discover and correct unwanted and unexpected access path changes before you are notified about them.

Related tasks

Preparing an application to run on Db2 for z/OS (Db2 Application programming and SQL)

Application programming default values

Each Db2 for z/OS subsystem has a set of *application programming default values*. These values are the default settings for SQL applications that run on that subsystem. For example, the ENSCHEME application programming default value specifies the default encoding scheme for new objects. Some users colloquially refer to these values as *DECP values*.

System administrators generally set the application programming default values when they install or migrate Db2. After installation or migration, a system administrator should change these values only if

necessary and according to the update advice that is documented for each application programming value. Changing an application programming default value requires Db2 to be stopped and restarted.

For a list of application programming defaults, see Directory of subsystem parameters, panel fields, and application default values (Db2 Installation and Migration).

Application programming default values are contained in the DSNHDECP load module that is generated by installation job DSNTIJUZ. In a data sharing environment, each member is not required to have a separate application defaults load module. The DSNHDECP module is also required by the Db2 precompiler and coprocessor for C, C++, COBOL, and PL/I.

One way to check the application programming default values is to call stored procedure SYSPROC.ADMIN_INFO_SYSPARM by running sample job DSNTEJ6Z. For certain application programming defaults, you can also check the value by invoking the GETVARIABLE function.

Related concepts

Subsystem parameters
Subsystem parameters are settings that apply to a Db2 for z/OS subsystem. For example, the AUTH subsystem parameter controls whether Db2 checks authorizations. Many users colloquially refer to subsystem parameters as *zparms*.

Subsystem parameters and application defaults in a data sharing environment (Db2 Installation and Migration)

Related tasks

Tailoring Db2 11 installation and migration jobs with the CLIST (Db2 Installation and Migration)

Related reference

Directory of subsystem parameters, panel fields, and application default values (Db2 Installation and Migration)

DSNTIPF: Application programming defaults panel 1 (Db2 Installation and Migration)

Static SQL applications

For most Db2 users, static SQL provides a straightforward, efficient path to Db2 data.

The source form of a static SQL statement is embedded within an application program that is written in a traditional programming language such as C. The statement is prepared before the program is executed, and the operational form of the statement persists beyond the execution of the program. You can use static SQL when you know before run time what SQL statements your application needs to run.

When you use static SQL, you cannot change the form of SQL statements unless you make changes to the program. However, you can increase the flexibility of those statements by using host variables. Using static SQL and host variables is more secure than using dynamic SQL.

GUPI

Example: Assume that you are coding static SQL in a COBOL program. The following UPDATE statement can update the salary of any employee. When you write your program, you know that salaries must be updated, but you do not know until run time whose salaries should be updated, and by how much.

```
01  IOAREA.
    02  EMPID              PIC X(06).
    02  NEW-SALARY         PIC S9(7)V9(2) COMP-3.
⋮ (Other declarations)
READ CARDIN RECORD INTO IOAREA
  AT END MOVE 'N' TO INPUT-SWITCH.
⋮ (Other COBOL statements)
EXEC SQL
  UPDATE EMP
    SET SALARY = :NEW-SALARY
    WHERE EMPNO = :EMPID
END-EXEC.
```

GUPI

The UPDATE statement does not change, nor does its basic structure, but the input can change the results of the UPDATE statement.

Basic SQL coding concepts apply to traditional programming languages: C, C++, COBOL, Fortran, PL/I, and Assembler.

Suppose that you are writing an application program to access data in a Db2 database. When your program executes an SQL statement, the program needs to communicate with Db2. When Db2 finishes processing an SQL statement, Db2 sends back a return code, called the SQL *return code*. Your program should test the return code to examine the results of the operation.

Unique instructions and details apply to each language.

Related concepts

Submitting SQL statements to Db2
You can use several different methods to send SQL statements to Db2 in several ways.

Static SQL
The source form of a *static SQL statement* is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

Related reference

Developing high-performance data access applications with pureQuery

Declaration of table and view definitions

Declaring table or view definitions is optional, but they offer several advantages. You can declare a table or view by including an SQL DECLARE statement in your program.

Before your program issues SQL statements that retrieve, update, delete, or insert data, you must declare the tables and views that your program accesses. Declaring tables or views is not required; however, declaring them offers advantages such as documenting application programs and providing the precompiler with information that is used to check your embedded SQL statements.

GUPI

Example: The DECLARE TABLE statement (written in COBOL) for the DEPT table looks like the following example:

```
EXEC SQL
  DECLARE DEPT TABLE
    (DEPTNO    CHAR(3)           NOT NULL,
     DEPTNAME  VARCHAR(36)       NOT NULL,
     MGRNO     CHAR(6)                    ,
     ADMRDEPT  CHAR(3)           NOT NULL )
END-EXEC.
```

GUPI

For each traditional language, you delimit an SQL statement in your program between EXEC SQL and a statement terminator. In the preceding example, the EXEC SQL and END-EXEC delimit the SQL statement in a COBOL program.

As an alternative to coding the DECLARE statement yourself, you can use the Db2 subcomponent DCLGEN, the declarations generator.

Related reference

DECLARE STATEMENT (Db2 SQL)

Data access with host variables

You can use host variables, host variable arrays, and host structures in your application program to exchange data between the application and the DBMS.

A *host variable* is a data item that you declare in a program for use within an SQL statement. You can:

- Retrieve data into the host variable for your application program's use.
- Place data into the host variable to insert into a table or to change the contents of a row.
- Use the data in the host variable when evaluating a WHERE or HAVING clause.
- Assign the value in the host variable to a special register. A *special register* is a storage area that Db2 defines for a process to hold information that SQL statements can reference.

  Example 1: The CURRENT SQLID special register contains the SQL authorization ID of a process, which is set in an SQL statement. Db2 replaces the register name with the value of the authorization ID when the SQL statement runs.
- Use the host variable to indicate a null value

How you code a host variable varies according to the programming language that you use. Some languages require a separate declaration section for SQL variables. In this case, you can code the BEGIN and END DECLARE SECTION statements in an application program wherever variable declarations can appear according to the rules of the host language. A host variable declaration section starts with the BEGIN DECLARE SECTION statement and ends with the END DECLARE SECTION statement. The host variable must be preceded with a *:hostvar*

The INTO clause of the SELECT statement names one or more host variables to contain the returned column values. For host variables and host variable arrays, the named variables correspond one-to-one with the list of column names in the SELECT list.

The example that follows uses a host variable to retrieve a single row of data.

GUPI

Example 2: Suppose that you want to retrieve the EMPNO, LASTNAME, and DEPT column values from a single row in the EMP table. You can define a host variable in your program to hold each column. The host variable consists of the local variable name, preceded by a colon. You then can name the data areas with an INTO clause, as shown:

```
EXEC SQL
  SELECT EMPNO, LASTNAME, DEPT
    INTO :CBLEMPNO, :CBLNAME, :CBLDEPT
    FROM EMP
    WHERE EMPNO = :EMPID
END-EXEC.
```

GUPI

You must declare the host variables CBLEMPNO, CBLNAME, and CBLDEPT in the data declaration portion of the program. The data types of the host variables must be compatible with the SQL data types of the columns EMPNO, LASTNAME, and DEPT of the EMP table.

Suppose that you don't know how many rows Db2 will return, or you expect more than one row to return. In either case, you must use an alternative to the SELECT ... INTO statement. Using a Db2 cursor, an application can process a set of rows and retrieve rows from the result table.

Related concepts

Row retrieval with a cursor
Db2 has a mechanism called a *cursor*. Using a cursor is like keeping your finger on a particular line of text on a printed page.

Host variables (Db2 Application programming and SQL)

Host variables (Db2 SQL)

Data access with host variable arrays

A *host variable array* is a data array that is declared in a host language for use within an SQL statement. You can retrieve data into host variable arrays for use by your application program. You can also place data into host variable arrays to insert rows into a table.

You can specify host variable arrays in C, C++, COBOL, or PL/I. Each host variable array contains values for a column, and each element of the array corresponds to a value for a column. You must declare the array in the host program before you use it.

GUPI

Example: The following statement uses the main host variable array, COL1, and the corresponding indicator array, COL1IND. Assume that COL1 has 10 elements. The first element in the array corresponds to the first value, and so on. COL1IND must have at least 10 entries.

```
EXEC SQL
  SQL FETCH FIRST ROWSET FROM C1 FOR 5 ROWS
    INTO :COL1 :COL1IND
END-EXEC.
```

GUPI

Related concepts

Host variables (Db2 Application programming and SQL)

Data access with host structures

A *host structure* is a group of host variables that an SQL statement can refer to by using a single name. When the host language environment allows it, you can use host language statements to define the host structures.

GUPI

Example 1: Assume that your COBOL program includes the following SQL statement:

```
EXEC SQL
  SELECT EMPNO, FIRSTNME, LASTNAME, DEPT
    INTO :EMPNO, :FIRSTNME, :LASTNAME, :WORKDEPT
    FROM VEMP
    WHERE EMPNO = :EMPID
END-EXEC.
```

Now assume that you want to avoid listing the host variables in the preceding example.

Example 2: You can substitute the name of a structure, such as :PEMP, that contains :EMPNO, :FIRSTNME, :LASTNAME, and :DEPT:

```
EXEC SQL
  SELECT EMPNO, FIRSTNME, LASTNAME, WORKDEPT
    INTO :PEMP
    FROM VEMP
    WHERE EMPNO = :EMPID
END-EXEC.
```

GUPI

You can declare a host structure in your program. You can also use DCLGEN to generate a COBOL record description, PL/I structure declaration, or C structure declaration that corresponds to the columns of a table.

Related concepts

Host variables (Db2 Application programming and SQL)

Row retrieval with a cursor

Db2 has a mechanism called a *cursor*. Using a cursor is like keeping your finger on a particular line of text on a printed page.

In Db2, an application program uses a cursor to point to one or more rows in a set of rows that are retrieved from a table. You can also use a cursor to retrieve rows from a result set that is returned by a stored procedure. Your application program can use a cursor to retrieve rows from a table.

You can retrieve and process a set of rows that satisfy the search condition of an SQL statement. When you use a program to select the rows, the program processes one or more rows at a time.

The SELECT statement must be within a DECLARE CURSOR statement and cannot include an INTO clause. The DECLARE CURSOR statement defines and names the cursor, identifying the set of rows to retrieve with the SELECT statement of the cursor. This set of rows is referred to as the result table.

After the DECLARE CURSOR statement executes, you process the result table of a cursor as follows:

1. Open the cursor before you retrieve any rows.

   To tell Db2 that you are ready to process the first row of the result table, have your program issue the OPEN statement. Db2 then uses the SELECT statement within the DECLARE CURSOR statement to identify a set of rows. If you use host variables in that SELECT statement, Db2 uses the *current value* of the variables to select the rows.

2. Use a FETCH statement to retrieve one or more rows.

   The simplest form of the FETCH statement retrieves a single row of the result table by using a *row-positioned* cursor. At any point in time, a row-positioned cursor retrieves at most a single row from the result table into host variables. You can use a FETCH statement to retrieve more than one row of the result table by using a cursor that is enabled to process rowsets. A *rowset* is a set of rows that is retrieved through a multiple-row fetch.

   When your program issues a row-positioned FETCH statement, Db2 uses the cursor to point to a row in the result table, making it the current row. Db2 then moves the current row contents into the program host variables that you specified in the INTO clause of the FETCH statement. The FETCH statement moves the cursor. You can use host variable arrays and return multiple rows of data with a single FETCH statement.

3. Close the cursor when the end-of-data condition occurs.

   If you finish processing the rows of the result table and you want to use the cursor again, issue a CLOSE statement to close the cursor.

Recommendation: Explicitly close the cursor when you finish using it.

Your program can have several cursors. Each cursor has the following requirements:

- DECLARE CURSOR statement to define the cursor
- OPEN and CLOSE statements to open and close the cursor
- FETCH statement to retrieve rows from the result table of the cursor

You must declare host variables before you refer to them in a DECLARE CURSOR statement. To define and identify a set of rows that are to be accessed with a cursor, issue a DECLARE CURSOR statement. The DECLARE CURSOR statement names a cursor and specifies a SELECT statement. The SELECT statement defines the criteria for the rows that belong in the result table.

You can use cursors to fetch, update, or delete one or more rows of a table, but you cannot use them to insert a row into a table.

Suppose that your program examines data about people in department D11 and keeps the data in the EMP table. The following examples show the SQL statements that you must include in a COBOL program to define and use a cursor. In these examples, the program uses the cursor to process a set of rows from the EMP table.

GUPI

Example: Define the cursor: The following statement defines a cursor named THISEMP:

```
EXEC SQL
  DECLARE THISEMP CURSOR FOR
    SELECT EMPNO, LASTNAME,
    DEPT, JOB
    FROM EMP
    WHERE DEPT = 'D11'
  FOR UPDATE OF JOB
END-EXEC.
```

Example: Open the cursor: The following statement opens the cursor:

```
EXEC SQL
  OPEN THISEMP
END-EXEC.
```

Example: Use the cursor to retrieve a row: The following statement uses the cursor, THISEMP, to retrieve a row:

```
EXEC SQL
  FETCH THISEMP
  INTO :EMP-NUM, :NAME2,
  :DEPT, :JOB-NAME
END-EXEC.
```

Example: Update the current row using the cursor: The following statement uses the cursor, THISEMP, to update the JOB value for specific employees in department D11:

```
EXEC SQL
  UPDATE EMP
  SET JOB = :NEW-JOB
  WHERE CURRENT OF THISEMP
END-EXEC.
```

Example: Close the cursor: The following statement closes the cursor:

```
EXEC SQL
  CLOSE THISEMP
END-EXEC.
```

GUPI

If the cursor is not scrollable, each fetch positions the cursor at the next sequential row, or set of rows. A *scrollable cursor* can scroll forward and backward, and can be repositioned at the beginning, at the end, or at a relative offset point. Applications can use a powerful set of SQL statements to fetch data by using a cursor in random order. Scrollable cursors are especially useful for screen-based applications. You can specify that the data in the result table is to remain static. For example, an accounting application can require that data is to remain constant, whereas an airline reservation system application must display the latest flight availability information.

You can also define options on the DECLARE CURSOR statement that specify how sensitive a scrollable cursor is to changes in the underlying data when inserts, updates, or deletes occur.

- A *sensitive cursor* is sensitive to changes that are made to the database after the result table is generated. For example, when an application executes positioned UPDATE and DELETE statements with the cursor, those changes are visible in the result table.
- An *insensitive cursor* is not sensitive to inserts, updates, or deletes that are made to the underlying rows of a result table after the result table is generated. For example, the order of the rows and the values for each row of the result table do not change after the application opens the cursor.

To indicate that a cursor is scrollable, you declare it with the SCROLL keyword.

GUPI

Example: The following example shows a declaration for an insensitive scrollable cursor:

```
EXEC SQL DECLARE C1 INSENSITIVE SCROLL CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DEPT
  ORDER BY DEPTNO
END-EXEC.
```

To use this cursor to fetch the fifth row of the result table, you can use a FETCH statement like the following example:

```
EXEC SQL FETCH ABSOLUTE +5 C1 INTO :HVDEPTNO, :DEPTNAME, :MGRNO;
```

GUPI

Db2 for z/OS provides another type of cursor called a *dynamic scrollable cursor*. With a dynamic scrollable cursor, applications can scroll directly on a base table while accessing the most current data.

Related reference

DECLARE CURSOR (Db2 SQL)

FETCH (Db2 SQL)

Ways to check the execution of SQL statements

Db2 offers several ways to check the execution of SQL statements in an program.

A program that includes SQL statements can have an area that is set apart for communication with Db2—an *SQL communication area (SQLCA)*. When Db2 processes an SQL statement in your program, it places return codes in the SQLSTATE and SQLCODE host variables or in corresponding fields of the SQLCA. The return codes indicate whether the statement executed successfully or failed.

Recommendation: Because the SQLCA is a valuable problem-diagnosis tool, include the necessary instructions to display some of the information that is in the SQLCA in your application programs.

You can use a GET DIAGNOSTICS statement or a WHENEVER statement in your program to supplement checking SQLCA fields after each SQL statement runs.

- The GET DIAGNOSTICS statement returns diagnostic information about the last SQL statement that was executed. You can request specific types of diagnostic information or all available diagnostic information about a statement. For example, the GET DIAGNOSTICS statement returns the number of rows that are affected by a data insert, update, or delete.
- The WHENEVER statement allows you to specify what to do if a general condition is true. Db2 checks the SQLCA and continues processing your program. If an error, exception, or warning results when an SQL statement is executed, Db2 branches to another area in your program. The program can then examine the SQLSTATE or SQLCODE to react specifically to the error or exception.

Related reference

GET DIAGNOSTICS (Db2 SQL)

WHENEVER (Db2 SQL)

Dynamic SQL applications

With dynamic SQL, Db2 prepares and executes the SQL statements within a program while the program is running. Dynamic SQL is a good choice when you do not know the format of an SQL statement before you write or run a program.

An application that uses dynamic SQL generates an SQL statement in the form of a character string or accepts an SQL statement as input. Depending on the needs of the application, you might be able to simplify the programming. Try to plan the application so that it does not use SELECT statements, or so that it uses only those statements that return a known number of values of known data types. In general, more complex dynamic programs are those in which you do not know in advance about the SQL statements that the application issues.

A typical dynamic SQL application takes the following steps:

1. Translates the input data into an SQL statement.
2. Prepares the SQL statement to execute and acquires a description of the result table (if any).
3. Obtains, for SELECT statements, enough main storage to contain retrieved data.
4. Executes the statement or fetches the rows of data.
5. Processes the returned information.
6. Handles SQL return codes.

GUPI This example shows a portion of a C program that dynamically issues SQL statements to Db2. Assume that you are writing a program to keep an inventory of books. The table that you need to update depends on input to your program. This example shows how you can build an SQL statement and then call Db2 to execute it.

```
/*********************************************************/
/* Determine which table to update, then build SQL       */
/* statement dynamically into 'stmt' variable.           */
/*********************************************************/
  strcpy(stmt,"UPDATE ");

  EXEC SQL SELECT TYPE INTO :book_type FROM BOOK_TYPES WHERE
  TITLE=:bktitle;

  IF (book_type=='FICTION') strcpy(table_name,"FICTION_BOOKS");
  ELSE strcpy(table_name,"NON_FICTION_BOOKS");

  strcat(stmt,table_name);
  strcat(stmt,
  " SET INVENTORY = INVENTORY-1 WHERE TITLE = :bktitle");
/*********************************************************/
/* PREPARE and EXECUTE the statement                     */
/*********************************************************/
EXEC SQL PREPARE OBJSTMT FROM :stmt;
EXEC SQL EXECUTE OBJSTMT;
```

GUPI

Related concepts

Submitting SQL statements to Db2
You can use several different methods to send SQL statements to Db2 in several ways.

Differences between static and dynamic SQL (Db2 Application programming and SQL)

Use of Java to execute static and dynamic SQL
Db2 for z/OS supports SQLJ and JDBC. In general, Java applications use SQLJ for static SQL, JDBC for dynamic SQL, and PureQuery for both static and dynamic SQL.

How authorization IDs control data access
One of the ways that Db2 controls access to data is through the use of identifiers. A set of one or more Db2 identifiers, called *authorization IDs*, represents every process that connects to or signs on to Db2.

Differences between Db2 ODBC and embedded SQL (Db2 Programming for ODBC)

Related tasks

Including dynamic SQL in your program (Db2 Application programming and SQL)

Improving dynamic SQL performance (Db2 Performance)

Use of ODBC to execute dynamic SQL

Open Database Connectivity (ODBC) lets you access data through ODBC function calls in your application. The ODBC interface eliminates the need for precompiling and binding your application and increases the portability of your application.

The ODBC interface is specifically designed for C and C++ applications to access relational databases. Applications that use the ODBC interface might be executed on a variety of data sources without being compiled against each of the databases. ODBC ideally suits the client/server environment in which the target data source might be unknown when the application is built.

You execute SQL statements by passing them to Db2 through an ODBC function call. The function calls allow an application to connect to the data source, issue SQL statements, and receive returned data and status information.

You can prepare an SQL statement by calling the ODBC SQLPrepare() function. You then execute the statement by calling the ODBC SQLExecute() function. In both cases, the application does not contain an embedded PREPARE or EXECUTE statement. You can execute the statement, without preparation, by passing the statement to the ODBC SQLExecDirect() function.

Another advantage of ODBC access is that it can help hide the differences between system catalogs of different database servers. Unlike embedded SQL, Db2 ODBC provides a consistent interface for applications to query and retrieve system catalog information across the Db2 Database family of database management systems. This capability reduces the need to write catalog queries that are specific to each database server. Db2 ODBC can return result tables to those programs.

GUPI

Example:

This example shows a portion of an ODBC program for keeping an inventory of books.

```
/*********************************************************/
/* Determine which table to update                       */
/*********************************************************/
rc = SQLBindParameter( hStmt,
                       1,
                       SQL_PARAM_INPUT,
                       SQL_C_CHAR,
                       SQL_CHAR,
                       50,
                       0,
                       bktitle,
                       sizeof(bktitle),
                       &bktitle_len);
if( rc != SQL_SUCCESS ) goto dberror;

rc = SQLExecDirect( hStmt,
                           "SELECT TYPE FROM BOOK_TYPES WHERE TITLE=?"
                           SQL_NTS );
if( rc != SQL_SUCCESS ) goto dberror;

rc = SQLBindCol( hStmt,
                 1,
                 SQL_C_CHAR,
                 book_type,
                 sizeof(book_type),
                 &book_type_len);
if( rc != SQL_SUCCESS ) goto dberror;

rc = SQLFetch( hStmt );
if( rc != SQL_SUCCESS ) goto dberror;

rc = SQLCloseCursor( hStmt );
if( rc != SQL_SUCCESS ) goto dberror;
/*********************************************************/
/* Update table                                          */
/*********************************************************/
strcpy( (char *)update_sqlstmt, (char *)"UPDATE ");
if( strcmp( (char *)book_type, (char *)"FICTION") == 0)
{
    strcat( (char *)update_sqlstmt, (char *)"FICTION_BOOKS" );
}
else
{
    strcpy( (char *)update_sqlstmt, (char *)"NON_FICTION_BOOKS" );
}
strcat( (char *)update_sqlstmt,
        (char *)" SET INVENTORY = INVENTORY-1 WHERE TITLE = ?");

rc = SQLPrepare( hStmt, update_sqlstmt, SQL_NTS );
if( rc != SQL_SUCCESS ) goto dberror;

rc = SQLExecute( hStmt );
if( rc != SQL_SUCCESS ) goto dberror;
```

```
rc = SQLEndTran( SQL_HANDLE_DBC, hDbc, SQL_COMMIT );
if( rc != SQL_SUCCESS ) goto dberror;
```

GUPI

Related concepts

Dynamic SQL applications

With dynamic SQL, Db2 prepares and executes the SQL statements within a program while the program is running. Dynamic SQL is a good choice when you do not know the format of an SQL statement before you write or run a program.

Related tasks

Enabling dynamic SQL statement caching for ODBC function calls (Db2 Programming for ODBC)

Use of Java to execute static and dynamic SQL

Db2 for z/OS supports SQLJ and JDBC. In general, Java applications use SQLJ for static SQL, JDBC for dynamic SQL, and PureQuery for both static and dynamic SQL.

By using the Java programming language you gain the following key advantages:

- You can write an application on any Java-enabled platform and run it on any platform to which the Java Development Kit (JDK) is ported.
- You can develop an application once and run it anywhere, which offers the following potential benefits:
  - Reduced development costs
  - Reduced maintenance costs
  - Reduced systems managements costs
  - Flexibility in supporting diverse hardware and software configurations

The following table shows some of the major differences between SQLJ and JDBC.

Table 35. Comparison of SQLJ and JDBC

SQLJ characteristics	JDBC characteristics
SQLJ follows the static SQL model and offers performance advantages over JDBC.	JDBC follows the dynamic SQL model.
SQLJ source programs are smaller than equivalent JDBC programs because SQLJ automatically generates certain code that developers must include in JDBC programs.	JDBC source programs are larger than equivalent SQLJ programs because certain code that the developer must include in JDBC programs is generated automatically by SQLJ.
SQLJ checks data-types during the program preparation process and enforces strong typing between table columns and Java host expressions.	JDBC passes values to and from SQL tables without checking data types at compile time.
In SQLJ programs, you can embed Java host expressions in SQL statements.	JDBC requires a separate statement for each bind variable and specifies the binding by position number.
SQLJ provides the advantages of static SQL authorization checking. With SQLJ, the authorization ID under which SQL statements run is the plan or package owner. Db2 checks the table privileges at bind time.	Because JDBC uses dynamic SQL, the authorization ID under which SQL statements run is not known until run time, so no authorization checking of table privileges can occur until run time.

SQLJ support

Db2 for z/OS includes SQLJ, which provides support for embedding static SQL statements in Java applications and servlets. *Servlets* are application programs that are written in Java and that run on a web server.

Because SQLJ coexists with JDBC, an application program can create a JDBC connection and then use that connection to run dynamic SQL statements through JDBC and embedded static SQL statements through SQLJ.

A group of companies that includes Oracle, Hewlett Packard, and IBM, initially developed SQLJ to complement the dynamic SQL JDBC model with a static SQL model.

The SQLJ coding to update the salary of any employee is as follows:

GUPI

```
#sql [myConnCtxt] { UPDATE EMP
                            SET SALARY = :newSalary
                            WHERE EMPNO = :empID };
```

GUPI

By using SQLJ you gain the following advantages:

- Portable applications across platforms and database management systems.
- Strong typing, with compile and bind-time checking to ensure that applications are well designed for the database.
- Superior performance, manageability, and authorization checking of static SQL.
- Improved programmer productivity and easier maintenance. In comparison to a JDBC application, the resulting program is typically shorter and easier to understand.
- Familiarity for programmers who use embedded SQL in other traditional programming languages.

Related concepts

Db2 database design

After building a logical design and physical design of your relational database and collecting the processing requirements, you can move to the implementation stage. In general, implementing your physical design involves defining the various objects and enforcing the constraints on the data relationships.

JDBC support

Db2 for z/OS supports applications that use Sun Microsystems JDBC interfaces to access Db2 data by using dynamic SQL. Support for JDBC enables organizations to write Java applications that access local Db2 data, and access remote relational data on a server that supports DRDA.

Sun Microsystems developed the JDBC specifications. The JDBC specifications define a set of APIs (based on ODBC) that allow Java applications to access relational data. The APIs provide a generic interface for writing applications that run on multiple platforms and can access any SQL database. The APIs are defined within 16 classes, and they support basic SQL functions for connecting to a database, running SQL statements, and processing results. Together, these interfaces and classes represent the JDBC capabilities by which a Java application can access relational data.

GUPI

This example shows a portion of a JDBC program for that keeps an inventory of books.

```
/*******************************************************/
/* Determine which table to update, then build SQL     */
/* statement dynamically.                              */
/*******************************************************/
String tableName = null;
Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT TYPE FROM " +
```

```
" BOOK_TYPES WHERE " +
" TITLE = \"" + bkTitle + "\"");
if (rs.next())
{
  if (rs.getString(1).equalsIgnoreCase("FICTION"))
    tableName = "FICTION_BOOKS";
  else
    tableName = "NON_FICTION_BOOKS";
/*******************************************************/
/* PREPARE and EXECUTE the statement                   */
/*******************************************************/
stmt.executeUpdate("UPDATE " + tableName + " SET INVENTORY = INVENTORY-1 " +
"WHERE TITLE = \"" + bkTitle + "\"");
}
rs.close();
stmt.close();
```

GUPI

Db2 for z/OS support for JDBC offers a number of advantages for accessing Db2 data:

- JDBC combines the benefit of running your applications in a z/OS environment with the portability and ease of writing Java applications.
- The JDBC interface offers the ability to change between drivers and access various databases without recoding your Java program.
- JDBC applications do not require precompiles or binds.
- JDBC provides a consistent interface for applications to query and retrieve system catalog information across the Db2 Database family of database management systems. This capability reduces the need to write catalog queries that are specific to each database server.

Related concepts

Submitting SQL statements to Db2
You can use several different methods to send SQL statements to Db2 in several ways.

Dynamic SQL applications
With dynamic SQL, Db2 prepares and executes the SQL statements within a program while the program is running. Dynamic SQL is a good choice when you do not know the format of an SQL statement before you write or run a program.

Use of ODBC to execute dynamic SQL
Open Database Connectivity (ODBC) lets you access data through ODBC function calls in your application. The ODBC interface eliminates the need for precompiling and binding your application and increases the portability of your application.

Related tasks

Including dynamic SQL in your program (Db2 Application programming and SQL)

Use of an application program as a stored procedure

A stored procedure is a compiled program that can execute SQL statements.

Stored procedures are stored at theDb2 local or remote server where they run. A typical stored procedure contains two or more SQL statements and some manipulative or logical processing in a program. A client application program uses the SQL CALL statement to invoke the stored procedure.

Consider using stored procedures for a client/server application that does at least one of the following things:

- Executes multiple remote SQL statements.

  Remote SQL statements can result in several send and receive operations across the network, which increases processor costs and elapsed times.

  Stored procedures can encapsulate many SQL statements into a single message to the Db2 server. The network traffic of stored procedures is a single send and receive operation for a series of SQL statements.

Locks on Db2 tables are not held across network transmissions, which reduces contention for resources at the server.

- Accesses tables from a dynamic SQL environment in which table privileges for the application that is running are undesirable.

  Stored procedures allow static SQL authorization from a dynamic environment.

- Accesses host variables for which you want to check security and integrity.

  Stored procedures remove SQL applications from the workstation, preventing workstation users from manipulating the contents of sensitive SQL statements and host variables.

- Creates a result set of rows to return to the client application.

Related concepts

Stored procedures (Db2 Application programming and SQL)

Related tasks

Implementing Db2 stored procedures (Db2 Administration Guide)

Languages used to create stored procedures

Stored procedures can be written in a variety of programming languages from object-oriented programming languages to traditional programming languages.

You can write stored procedures in the following programming languages:

Java
If you have more experience writing applications in an object-oriented programming environment, you might want to create stored procedures by using Java

SQL procedural language
If your application consists entirely of SQL statements, some simple control flow logic, and no complex application logic, you might choose to create your stored procedures by using the SQL procedural language.

REXX
You can create stored procedures by using REXX programs that can contain dynamic SQL. DBAs and programmers generally use REXX for administrative tasks.

Traditional programming languages: C, C++, COBOL, PL/I, and Assembler
All traditional language programs must be designed to run using Language Environment®. COBOL and C++ stored procedures can contain object-oriented extensions.

The program that calls the stored procedure can be in any language that supports the SQL CALL statement. ODBC and JDBC applications can use an escape clause to pass a stored procedure call to Db2.

Related concepts

Use of Java to execute static and dynamic SQL
Db2 for z/OS supports SQLJ and JDBC. In general, Java applications use SQLJ for static SQL, JDBC for dynamic SQL, and PureQuery for both static and dynamic SQL.

Use of the SQL procedural language to create a stored procedure
With SQL procedural language, you can write stored procedures that consist entirely of SQL statements.

Stored procedure processing

There are several steps to stored procedure processing.

The following figure illustrates processing without stored procedures.

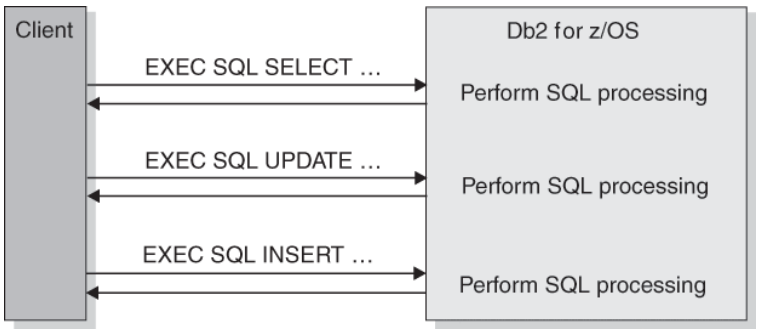


Figure 51. Processing without stored procedures

The following figure illustrates processing with stored procedures.

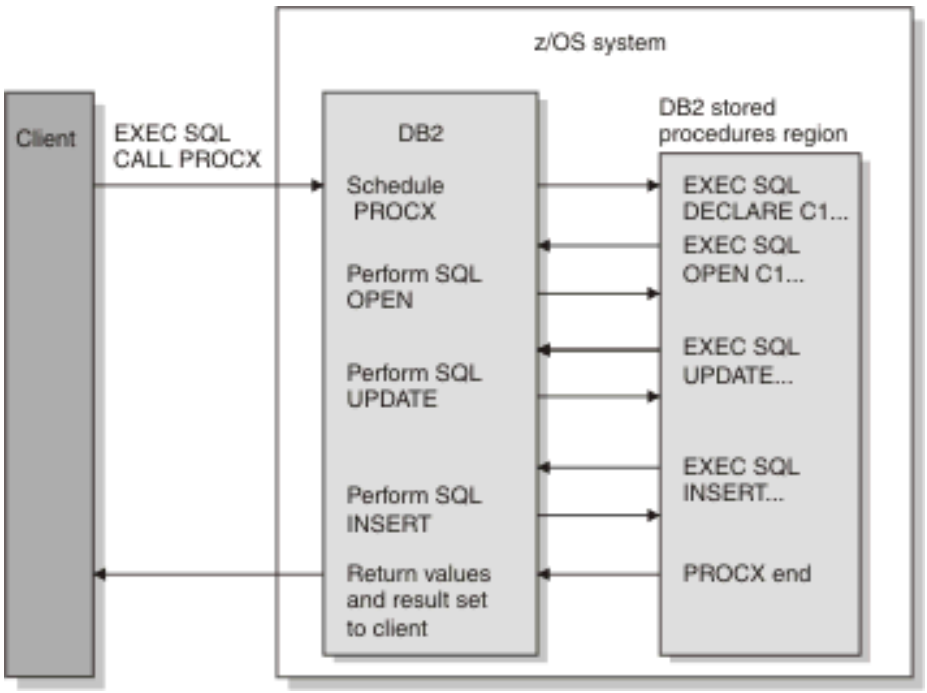


Figure notes:

- The workstation application uses the SQL CONNECT statement to create a conversation with Db2.
- Db2 creates a Db2 thread to process SQL requests. A *thread* is the Db2 structure that describes the connection of an application and traces application progress.
- The SQL statement CALL tells the Db2 server that the application is going to run a stored procedure. The calling application provides the necessary arguments.
- Db2 processes information about the request and loads the stored procedure program.
- The stored procedure executes SQL statements.

  One of the SQL statements opens a cursor that has been declared WITH RETURN. This action causes a result set to be returned to the workstation application.
- The stored procedure assigns values to the output parameters and exits. Control returns to the Db2 stored procedures region and goes from there to the Db2 subsystem.
- Control returns to the calling application, which receives the output parameters and the result set.

  The application can call other stored procedures, or it can execute additional SQL statements. Db2 receives and processes the COMMIT or ROLLBACK request. The commit or rollback operation covers all SQL operations that the application or the stored procedure executes during the unit of work, unless the procedure is defined with the AUTONOMOUS option. Autonomous procedures execute in a separate unit of work from the calling application.

  If the application involves IMS or CICS, similar processing occurs. This processing is based on the IMS or CICS synchronization model, rather than on an SQL COMMIT or ROLLBACK statement.

Figure 52. Processing with stored procedures

Related concepts

Stored procedures (Db2 Application programming and SQL)

Related tasks

Implementing Db2 stored procedures (Db2 Administration Guide)

Use of the SQL procedural language to create a stored procedure

With SQL procedural language, you can write stored procedures that consist entirely of SQL statements.

An SQL procedure can include declarations of variables, conditions, cursors, and handlers. The SQL procedure can also include flow control, assignment statements, and traditional SQL for defining and manipulating relational data. These extensions provide a procedural language for writing stored procedures, and they are consistent with the Persistent Stored Modules portion of the SQL standard.

Example: This example shows a simple SQL procedure (the syntax for the CREATE PROCEDURE statement shows only a portion of the statement clauses): GUPI

```
CREATE PROCEDURE ITERATOR()  LANGUAGE SQL
BEGIN
     ..
     DECLARE not_found CONDITION FOR SQLSTATE '02000';
     DECLARE c1 CURSOR FOR ....;
     DECLARE CONTINUE HANDLER FOR not_found                    (2)
            SET at_end = 1;
     OPEN c1;
     ftch_loop1: LOOP
         FETCH c1 INTO v_dept, v_deptname, v_admdept;          (1)
         IF at_end = 1 THEN
             LEAVE ftch_loop1;                                 (3)
         ELSEIF v_dept = 'D01' THEN
         INSERT INTO department (deptno, deptname, admrdept)
                VALUES ( 'NEW', v_deptname, v_admdept);
         END IF;
      END LOOP;
      CLOSE c1;
END
```

GUPI

In this example:

- Processing goes through `ftch_loop1`, assuming that a row is found.
- The first time that the FETCH does not find a row, processing goes to the HANDLER (1).
- The HANDLER sets the `at_end` flag. Because the procedure uses a CONTINUE HANDLER, processing continues at the next step after the FETCH (2).
- Processing continues with the CLOSE SQL statement (3).

Related concepts

Stored procedures (Db2 Application programming and SQL)

Related tasks

Implementing Db2 stored procedures (Db2 Administration Guide)

Use of development tools to create a stored procedure

Workstation-based development tools can help you create, install, and test stored procedures for Db2 for z/OS.

Stored procedures are portable across the entire family of Db2 servers including Db2 for z/OS, DB2 for i, and Db2 for Linux, UNIX, and Windows. If a Db2 subsystem is configured for the creation of SQL and Java stored procedures, you can create these stored procedures with the tools that are available in IBM Data Studio. These tools also provide steps for building and installing Db2 Java stored procedures on distributed systems. These tools also support read-only access to user-defined functions, triggers, tables, and views.

Related concepts

IBM Data Studio

IBM Data Studio is a set of powerful set of tools that help you manage enterprise data, databases, and data-driven applications.

Related reference

Working with stored procedures in IBM Data Studio (IBM Data Studio)

Create, test, and deploy a Db2 SQL procedure with Integrated Data Management Tools (IBM Data Studio)
Developing Db2 for z/OS stored procedures (IBM Data Studio)

Setup of the stored procedure environment

Setting up the stored procedure environment includes establishing the stored procedure environment and defining your stored procedure to Db2. Typically, a system administrator customizes the environment, and an application programmer defines the stored procedure.

Before a stored procedure can run, you must define it to Db2. Use the SQL CREATE PROCEDURE statement to define a stored procedure to Db2. To alter the definition, use the ALTER PROCEDURE statement.

Preparation of a stored procedure

You must consider several factors before you use a stored procedure.

A stored procedure can consist of more than one program, each with its own package. Your stored procedure can call other programs, stored procedures, or user-defined functions. Use the facilities of your programming language to call other programs.

If the stored procedure calls other programs that contain SQL statements, each of those called programs must have a Db2 package. The owner of the package or plan that contains the CALL statement must have EXECUTE authority for all packages that the other programs use.

When a stored procedure calls another program, Db2 determines which collection the program package belongs to.

Related tasks

Creating stored procedures (Db2 Application programming and SQL)

How applications can call stored procedures

You can use the SQL CALL statement to call a stored procedure and to pass a list of arguments to that procedure.

An application program can call a stored procedure in the following ways:

- Execute the CALL statement locally, or send the CALL statement to a server. The application executes a CONNECT statement to connect to the server. The application then executes the CALL statement, or it uses a three-part name to identify and implicitly connect to the server where the stored procedure is located.
- After connecting to a server, combine CALL statements with other SQL statements. To execute the CALL statement, you can either execute the CALL statement statically or use an escape clause in an ODBC or JDBC application to pass the CALL statement to Db2.

To execute a stored procedure, you need two types of authorization:

- Authorization to execute the stored procedure
- Authorization to execute the stored procedure package and any packages that are in the stored procedure package

If the owner of the stored procedure has authority to execute the packages, the person who executes the packages does not need the authority.

The authorizations that you need depend on whether the name of the stored procedure is explicitly specified on the CALL statement or is contained in a host variable.

If the stored procedure invokes user-defined functions or triggers, you need additional authorizations to execute the user-defined function, the trigger, and the user-defined function packages.

Related concepts

Example of a simple stored procedure (Db2 Application programming and SQL)

Related tasks

Calling a stored procedure from your application (Db2 Application programming and SQL)

Web application environment

Web-based applications run on a web application server and access data on an enterprise information system, such as a Db2 database server. The components of web-based applications are spread across multiple tiers, or layers.

This information describe the various components and architectural characteristics of web applications and the role that Db2 plays in the web application environment.

In general, the user interface is on the first tier, the application programs are on the middle tier, and the data sources that are available to the application programs are on the enterprise information system tier. Developing web-based applications across a multitiered architecture is referred to as *server-side programming*.

Writing server-side programs is complicated and requires a detailed understanding of web server interfaces. Fortunately, application servers, such as WebSphere Application Server, are available to simplify this task. Each of these application servers defines a development environment for web applications and provides a run time environment in which the web applications can execute. The application server code, which provides the run time environment, supports the appropriate interface for interacting with the web server. With application servers, you can write programs for the application server's run time environment. Developers of these programs can focus on the business logic of the web application, rather than on making the application work with a web server.

Components of web-based applications

All web-based database applications have three primary components: A web browser (or client), a web application server, and a database server.

Web-based database applications rely on a database server, which provides the data for the application. The database server sometimes also provides business logic in the form of stored procedures. Stored procedures can offer significant performance advantages, especially in a multi-tiered architecture. In addition to database servers, other enterprise information system components include IMS databases, WebSphere MQ messages, and CICS records.

The clients handle the presentation logic, which controls the way in which users interact with the application. In some cases, the client validates user-provided input. Web applications sometimes integrate Java applets into the client-side logic to improve the presentation layer.

Applet
A Java program that is part of a Hypertext Markup Language (HTML) page. (*HTML* is the standard method for presenting web data to users.) Applets work with Java-enabled browsers, such as Microsoft Internet Explorer; they are loaded when the HTML page is processed.

Web application servers manage the business logic. The business logic, typically written in Java, supports multitiered applications. The web application server can manage requests from a variety of remote clients. The web application layer might include JavaServer Pages (JSP) files, Java servlets, Enterprise JavaBeans (EJB) components, or web services.

JSP
A technology that provides a consistent way to extend web server functionality and create dynamic web content. The web applications that you develop with JSP technology are server and platform independent.

Servlet
A Java program that responds to client requests and generates responses dynamically.

EJB
A component architecture for building distributed applications with the Java programming model. Server transactional components are reusable and provide portability across application servers.

Web services
Self-contained, modular applications that provide an interface between the provider and the consumer of application resources. You can read more about web services later in this information.

Architectural characteristics of web-based applications

Some web-based applications use a two-tier architecture, and others use an *n*-tier architecture that consists of three or more tiers.

Two-tier architecture
In a two-tier architecture, the client is on the first tier. The database server and web application server reside on the same server machine, which is the second tier. This second tier serves the data and executes the business logic for the web application. Organizations that favor this architecture typically prefer to consolidate their application capabilities and database server capabilities on a single tier. The second tier is responsible for providing the availability, scalability, and performance characteristics for the organization's web environment.

***n*-tier architecture**
In an *n*-tier architecture, application objects are distributed across multiple logical tiers, typically three or four.

In a three-tier architecture, the database server does not share a server machine with the web application server. The client is on the first tier, as it is in a two-tier architecture. On the third tier, the database server serves the data. For performance reasons, the database server typically uses stored procedures to handle some of the business logic. The application server resides on the second tier. The application server handles the portion of the business logic that does not require the functionality that the database server provides. In this approach, hardware and software components of the second and third tiers share responsibility for the availability, scalability, and performance characteristics of the web environment.

In a four-tier architecture, more than one logical tier can exist within the middle tier or within the enterprise information system tier. For example:

- The middle tier might consist of more than one web server. Alternatively, an intermediate firewall can separate the web server from the application server in the middle tier.
- A database server on tier three can be the data source for a web server on the middle tier, and another database server on tier four is the data source for a database server on tier three.

If you survey all the web applications that are available today, you would find many variations. For example, the database servers can run on various platforms, as can the clients. Designers of web applications use various tools, which affect how the applications work and how they look. Different companies choose different tools. The puzzle pieces that comprise one company's puzzles end up being different from the puzzles of other companies.

In many cases, the client and server for a web application are on different operating systems. The client, for example, can be on a workstation-based operating system, such as Windows XP or UNIX. The server for the application can also be on a workstation-based server, or it can be on an enterprise server, such as z/OS. The following figure shows the two-tier connectivity between a workstation-based client and both types of servers.

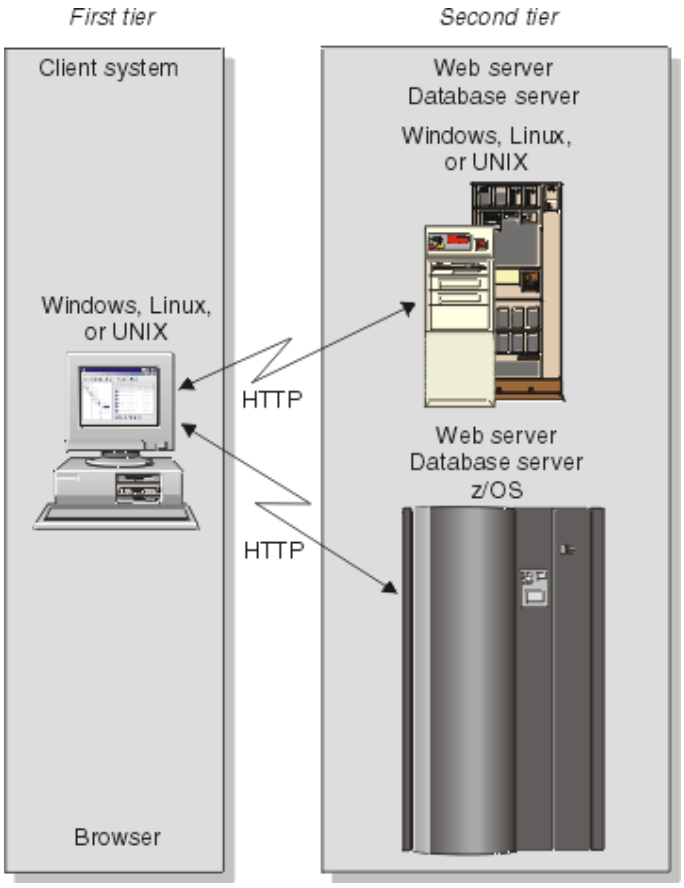


Figure 53. Two-tier connectivity between a workstation-based client and different database servers

The browser uses *Hypertext Transfer Protocol (HTTP)* to forward user requests to a second-tier server machine. (HTTP is a communication protocol that the web uses.) The web server on the second tier invokes the local database server to satisfy the data requirements of the application.

The following figure illustrates the use of an *n*-tier architecture. In this example, two web servers are installed on the middle tier: an HTTP server, such as the IBM HTTP Server, and a web application server, such as WebSphere Application Server. The application server supports the various components that might be running on the middle tier (JSP files, servlets, EJB, and web services). Each of these components performs functions that support client applications.

In the WebSphere Application Server environment, a device on tier one, such as a browser, can use HTTP to access the HTTP server on the middle tier. The HTTP server can then render output that is produced by JSPs, servlets, and other components that run in a WebSphere Application Server environment. The JSPs or servlets can use JDBC, SQLJ, or EJB (indirectly) to access data at a Db2 database server on the third tier.

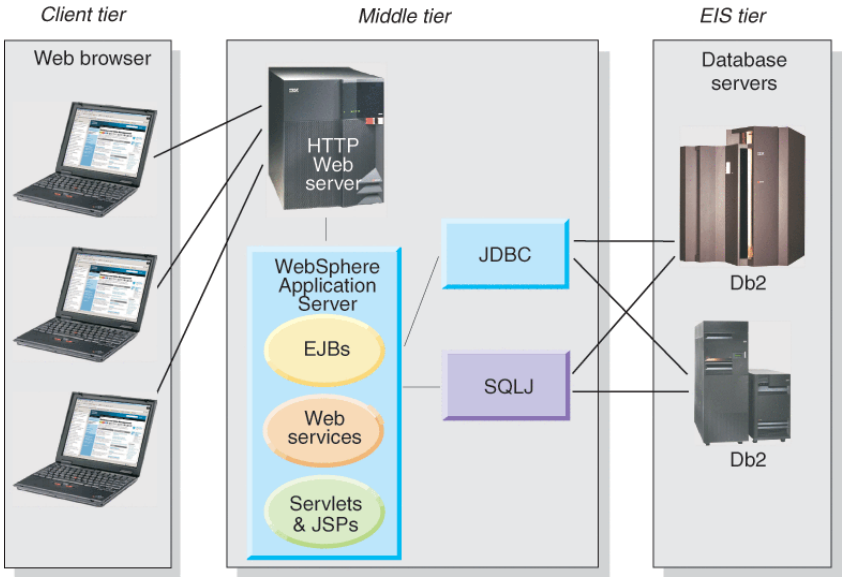


Figure 54. n-tier connectivity with a workstation-based client, two web servers, and different database servers

Benefits of Db2 for z/OS as a server

For each type of architecture, Db2 for z/OS offers a robust solution for web applications.

Specifically, using Db2 for z/OS as a database server for a web application provides the following advantages:

- **Exceptional scalability.** The volume of transactions on any web application varies. Transaction loads can increase, or spike, at different times of the day, on different days of the month, or at different times of the year. Transaction loads also tend to increase over time. In a Parallel Sysplex environment, Db2 for z/OS can handle the full range of transaction loads with little or no impact on performance. Any individual user is generally unaware of how busy the system is at a given point in time.
- **High degree of availability.** When Db2 for z/OS runs in a Parallel Sysplex environment, the availability of data and applications is very high. If one Db2 subsystem is unavailable, for example, because of maintenance, other Db2 subsystems in the Sysplex take over the workload. Users are unaware that part of the system is unavailable because they have access to the data and applications that they need.
- **Ability to manage a mixed workload.** Db2 for z/OS effectively and efficiently manages priorities of a mixed workload as a result of its tight integration with z/OS Workload Manager.
- **Protection of data integrity.** Users of Db2 for z/OS can benefit from the product's well-known strength in the areas of security and reliability.

Web-based applications and WebSphere Studio Application Developer

The WebSphere Studio Application Developer offers features that developers can use to create web-based applications.

WebSphere Studio Application Developer is designed for developers of Java and J2EE applications who require integrated web, XML, and web services support. This tool includes many built-in facilities and plug-ins that ease the task of accessing data stored in Db2 databases. (A *plug-in* is the smallest unit of function that can be independently developed and delivered.)

Each WebSphere Studio product offers the same integrated development environments and a common base of tools. Each product builds on the function of another product with additional plug-in tools. For example, WebSphere Studio Application Developer includes all WebSphere Studio Site Developer function plus plug-ins for additional function such as Enterprise JavaBeans support.

WebSphere Studio Site Developer
Offers a visual development environment that makes collaboration easy for web development teams.

WebSphere Studio Application Developer
Provides a development environment for developers of Java applications and adds tools for developing EJB applications.

WebSphere Studio Application Developer Integrated Edition
Includes WebSphere Studio Application Developer function and adds tools for integration with back-end systems.

WebSphere Studio Enterprise Developer
Includes WebSphere Studio Application Developer Integrated Edition function and additional function such as z/OS application development tools.

WebSphere Studio Application Developer provides an IDE for building, testing, debugging, and implementing many different components. Those components include databases, web, XML, and Java components. Java components include Java J2EE applications, JSP files, EJBs, servlets, and applets.

Because WebSphere Studio Application Developer is portable across operating systems, applications that you develop with WebSphere Studio Application Developer are highly scalable. This means that you can develop the applications on one system (such as AIX) and run them on much larger systems (such as z/OS).

WebSphere Studio Application Developer supports the Java 2 Enterprise Edition (J2EE) server model. J2EE is a set of specifications for working with multi-tiered applications on the J2EE platform. The J2EE platform includes services, APIs, and protocols for developing multi-tiered, web-based applications. The following figure shows a multi-tiered application development environment that supports web applications and J2EE applications.

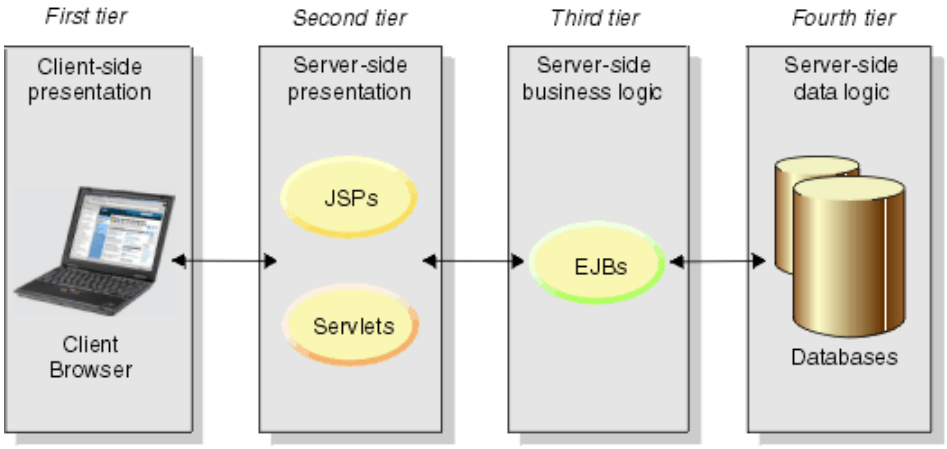


Figure 55. Web application development environment

Each WebSphere Studio product uses perspectives. A *perspective* is a set of views, editors, and tools that developers use to manipulate resources. You can use some of these perspectives to access Db2 databases.

Data perspective
Developers use the data perspective to manage the database definitions and connections that they need for application development. You can connect to Db2 databases and import database definitions, schemas, tables, stored procedures, SQL user-defined functions, and views. WebSphere Studio Application Developer provides an SQL editor that helps you create and modify SQL statements.

Using the data perspective, developers can create the following types of Db2 routines:

- SQL and Java stored procedures
- SQL user-defined functions
- User-defined functions that read or receive messages from WebSphere MQ message queues

When developers write stored procedures that use JDBC or SQL, they can then create a wrapper for the stored procedure as JavaBeans or as a method within a session EJB. Wrapping a stored procedure

avoids duplicating its business logic in other components and might result in a performance benefit. (A *wrapper* encapsulates an object and alters the interface or behavior of the object in some way. *Session beans* are enterprise beans that exist during one client/server session only.)

J2EE perspective

Developers work with the J2EE perspective to create EJB applications for accessing Db2. The J2EE perspective supports EJB 1.1 and EJB 2.0. This perspective provides graphical tools for viewing and editing Db2 schemas that help developers map entity EJBs to Db2 tables. *Entity beans* are enterprise beans that contain persistent data.

WebSphere Studio Application Developer also provides a feature that automatically generates a session EJB method to invoke a Db2 stored procedure.

Web perspective

Developers use the web perspective to generate web pages from SQL statements. WebSphere Studio Application Developer provides a tag library of JSP actions for database access. A *tag library* defines custom tags that are used throughout a document. Using the JSP tag libraries, developers can run SQL statements and stored procedures. They can easily update or delete the result sets that the SQL statements or stored procedures return.

Web services perspective

Developers use a built-in XML editor to create XML files for building Db2 web service applications based on SQL statements.

Related concepts

Tools and IDEs for developing Db2 applications
Application developers can use various application development tools and integrated development environments (IDEs) to create database applications, and stored procedures that access and use data in Db2 for z/OS.

Processing XML data with Db2 pureXML

You can use pureXML with your client applications to manage XML data in Db2 tables. You can store well-formed XML documents in their hierarchical form and retrieve all or portions of those documents.

Because the stored XML data is fully integrated into the Db2 database system, you can access and manage the XML data by using Db2 functions and capabilities. XML data is stored in a separate table space from SQL data types. The underlying storage of XML data is transparent to the application.

How is XML data stored?

The XML column data type is provided for storage of XML data in Db2 tables. With SQL statements you can:

- Create tables with XML columns.
- Add XML columns to existing tables.
- Create indexes over XML columns.
- Create triggers on tables with XML columns.
- Insert, update, or delete XML documents.

For details about the storage of XML data, see Storage structure for XML data (Db2 Programming for XML).

What is the pureXML data model?

The pureXML data model provides an abstract representation of one or more XML documents or fragments. The purpose of the data model is to define all permissible values of expressions. The pureXML data model is described in terms of sequences and items, atomic values, and nodes. For details about the pureXML data model, see pureXML data model (Db2 Programming for XML).

How is XML data represented in Db2?

The XML data type is used to define columns of a table that store XML values. This built-in data type enables you to store well-formed XML documents in a database. For details about the XML data type, see XML data type (Db2 Programming for XML).

How do you retrieve information from an XML document?

XQuery is a generalized language for interpreting, retrieving, and modifying XML data. XQuery, SQL, or a combination of both can be used to query and update XML data. SQL functions that return XML data or take XML arguments (referred to as SQL/XML functions) also enable XML data to be constructed or published from values that are retrieved from the database. The XMLTABLE SQL/XML function returns a table from the evaluation of XQuery expressions. XPath is a subset of XQuery and can be used to retrieve XML data with a simpler query syntax. For details about retrieving XML data, see Retrieving XML data (Db2 Programming for XML).

How can you update an XML document?

You can use SQL to update entire documents from XML columns, just as you retrieve data from any other type of column. You can also update entire or portions of documents with XQuery functions. For details about updating XML data, see Updates of XML columns (Db2 Programming for XML).

How can you improve query performance to XML data?

Indexing support is available for data that is stored in XML columns. The use of indexes over XML data can improve the efficiency of queries that you issue against XML documents. An XML index differs from a relational index in that a relational index applies to an entire column, whereas an XML index applies to part of the data in a column. You indicate which parts of an XML column are indexed by specifying an XML pattern, which is a limited XPath expression. For details about indexing XML data, see XML data indexing (Db2 Programming for XML).

What is XML schema validation?

An XML schema is used to validate and process XML documents. The XML schema repository (XSR) is a repository of XML schemas to process documents stored in XML columns with pureXML. For details about XML schema validation:

- See XML schema validation (Db2 Programming for XML)
- See XML schema management with the XML schema repository (XSR) (Db2 Programming for XML)

Which application development environments can you use?

Application development support of XML enables applications to combine XML and relational data access and storage. The following programming languages support the XML data type:

- Assembler
- C or C++ (embedded SQL or Db2 ODBC)
- COBOL
- Java (pureQuery, JDBC, or SQLJ)
- PL/I
- SQL PL

Can you use Db2 utilities with XML data?

Db2 for z/OS utilities support the XML data type. The storage structure for XML data and indexes is like the storage structure for LOB data and indexes. As with LOB data, XML data is not stored in the base table space, but it is stored in separate table spaces that contain only XML data. The XML table spaces also have their own index spaces. Therefore, the implications of using utilities for manipulating, backing up,

and restoring XML data and LOB data are similar. For details about how particular Db2 utilities interact with XML data, see XML support in Db2 utilities (Db2 Programming for XML).

Related information

Tutorial: Working with XML data (Db2 Programming for XML)

Chapter 6. Db2 database design

After building a logical design and physical design of your relational database and collecting the processing requirements, you can move to the implementation stage. In general, implementing your physical design involves defining the various objects and enforcing the constraints on the data relationships.

The objects in a relational database are organized into sets called *schemas*. A schema provides a logical classification of objects in the database. The schema name is used as the qualifier of SQL objects such as tables, views, indexes, and triggers.

This information explains the task of implementing your database design in a way that most new users will understand. When you actually perform the task, you might perform the steps in a different order.

You define, or create, objects by executing SQL statements. This information summarizes some of the naming conventions for the various objects that you can create. Also in this information, you will see examples of the basic SQL statements and keywords that you can use to create objects in a Db2 database. (This information does not document the complete SQL syntax.)

Tip: When you create Db2 objects (such as tables, table spaces, views, and indexes), you can precede the object name with a qualifier to distinguish it from objects that other people create. (For example, MYDB.TSPACE1 is a different table space than YOURDB.TSPACE1.) When you use a qualifier, avoid using SYS as the first three characters. If you do not specify a qualifier, Db2 assigns a qualifier for the object.

Related concepts

Db2 objects and their relationships
Logical data modeling and physical data modeling are two tasks that you need to perform to design a Db2 database.

Db2 objects and their relationships

Logical data modeling and physical data modeling are two tasks that you need to perform to design a Db2 database.

When you design any database, you need to answer many different questions. The same is true when you design a Db2 database. How will you organize your data? How will you create relationships between tables? How should you define the columns in your tables? What type of table space should you use?

To design a database, you perform two general tasks. The first task is logical data modeling, and the second task is physical data modeling. In logical data modeling, you design a model of the data without paying attention to specific functions and capabilities of the DBMS that stores the data. In fact, you could even build a logical data model without knowing which DBMS you will use. Next comes the task of physical data modeling. This is when you move closer to a physical implementation. The primary purpose of the physical design stage is to optimize performance while ensuring the integrity of the data.

This information begins with an introduction to the task of logical data modeling. The logical data modeling topic focuses on the entity-relationship model and provides an overview of the Unified Modeling Language (UML) and IBM Rational Data Architect. The information ends with the task of physical database design.

After completing the logical and physical design of your database, you implement the design.

Related concepts

Application of business rules to relationships
Whether a given relationship is one-to-one, one-to-many, many-to-one, or many-to-many, your relationships need to make good business sense.

Db2 database design
After building a logical design and physical design of your relational database and collecting the processing requirements, you can move to the implementation stage. In general, implementing your

© Copyright IBM Corp. 2001, 2020

physical design involves defining the various objects and enforcing the constraints on the data relationships.

Related tasks

Creating tables for data integrity (Db2 Application programming and SQL)

Altering a table for referential integrity (Db2 Administration Guide)

Logical database design using entity-relationship modeling

Before you implement a database, you should plan or design it so that it satisfies all requirements. This first task of designing a database is called logical design.

Related concepts

Logical database design with Unified Modeling Language

UML modeling is based on object-oriented programming principals. UML defines a standard set of modeling diagrams for all stages of developing a software system.

Physical database design

The physical design of your database optimizes performance while ensuring data integrity by avoiding unnecessary data redundancies. During physical design, you transform the entities into tables, the instances into rows, and the attributes into columns.

Data modeling

Logical data modeling is the process of documenting the comprehensive business information requirements in an accurate and consistent format.

Designing and implementing a successful database, one that satisfies the needs of an organization, requires a logical data model. Analysts who do data modeling define the data items and the business rules that affect those data items. The process of data modeling acknowledges that business data is a vital asset that the organization needs to understand and carefully manage.

Consider the following business facts that a manufacturing company needs to represent in its data model:

- Customers purchase products.
- Products consist of parts.
- Suppliers manufacture parts.
- Warehouses store parts.
- Transportation vehicles move the parts from suppliers to warehouses and then to manufacturers.

These are all business facts that a manufacturing company's logical data model needs to include. Many people inside and outside the company rely on information that is based on these facts. Many reports include data about these facts.

Any business, not just manufacturing companies, can benefit from the task of data modeling. Database systems that supply information to decision makers, customers, suppliers, and others are more successful if their foundation is a sound data model.

An overview of the data modeling process

You might wonder how people build data models. Data analysts can perform the task of data modeling in a variety of ways. (This process assumes that a data analyst is performing the steps, but some companies assign this task to other people in the organization.) Many data analysts follow these steps:

1. **Build critical user views.**

   Analysts begin building a logical data model by carefully examining a single business activity or function. They develop a *user view*, which is the model or representation of critical information that the business activity requires. (In a later stage, the analyst combines each individual user view with all the other user views into a consolidated logical data model.) This initial stage of the data modeling process is highly interactive. Because data analysts cannot fully understand all areas of the business that they are modeling, they work closely with the actual users. Working together, analysts and users define the

major *entities* (significant objects of interest) and determine the general relationships between these entities.

2. **Add key business rules to user views.**

   Next, analysts add key detailed information items and the most important business rules. Key business rules affect insert, update, and delete operations on the data.

   Example 1: A business rule might require that each customer entity have at least one unique identifier. Any attempt to insert or update a customer identifier that matches another customer identifier is not valid. In a data model, a unique identifier is called a primary key.

3. **Add detail to user views and validate them.**

   After the analysts work with users to define the key entities and relationships, they add other descriptive details that are less vital. They also associate these descriptive details, called *attributes*, to the entities.

   Example 2: A customer entity probably has an associated phone number. The phone number is a non-key attribute of the customer entity.

   Analysts also validate all the user views that they have developed. To validate the views, analysts use the normalization process and process models. *Process models* document the details of how the business will use the data. You can read more about process models and data models in other books on those subjects.

4. **Determine additional business rules that affect attributes.**

   Next, analysts clarify the data-driven business rules. *Data-driven business rules* are constraints on particular data values. These constraints need to be true, regardless of any particular processing requirements. Analysts define these constraints during the data design stage, rather than during application design. The advantage to defining data-driven business rules is that programmers of many applications don't need to write code to enforce these business rules.

   Example 3: Assume that a business rule requires that a customer entity have either a phone number or an address, or both. If this rule doesn't apply to the data itself, programmers must develop, test, and maintain applications that verify the existence of one of these attributes.

   Data-driven business requirements have a direct relationship with the data, thereby relieving programmers from extra work.

5. **Integrate user views.**

   In this last phase of data modeling, analysts combine the different user views that they have built into a consolidated logical data model. If other data models already exist in the organization, the analysts integrate the new data model with the existing one. At this stage, analysts also strive to make their data model flexible so that it can support the current business environment and possible future changes.

   Example 4: Assume that a retail company operates in a single country and that business plans include expansion to other countries. Armed with knowledge of these plans, analysts can build the model so that it is flexible enough to support expansion into other countries.

Recommendations for logical data modeling

To build sound data models, analysts follow a well-planned methodology, which includes:

- Working interactively with the users as much as possible.
- Using diagrams to represent as much of the logical data model as possible.
- Building a *data dictionary* to supplement the logical data model diagrams. (A data dictionary is a repository of information about an organization's application programs, databases, logical data models, users, and authorizations. A data dictionary can be manual or automated.)

Data modeling: Some practical examples

To perform the data modeling task, you begin by defining your entities, the significant objects of interest. Entities are the things about which you want to store information. For example, you might want to define an entity for employees called EMPLOYEE because you need to store information about everyone who works for your organization. You might also define an entity, called DEPARTMENT, for departments.

Next, you define primary keys for your entities. A primary key is a unique identifier for an entity. In the case of the EMPLOYEE entity, you probably need to store lots of information. However, most of this information (such as gender, birth date, address, and hire date) would not be a good choice for the primary key. In this case, you could choose a unique employee ID or number (EMPLOYEE_NUMBER) as the primary key. In the case of the DEPARTMENT entity, you could use a unique department number (DEPARTMENT_NUMBER) as the primary key.

After you decide on the entities and their primary keys, you can define the relationships that exist between the entities. The relationships are based on the primary keys. If you have an entity for EMPLOYEE and another entity for DEPARTMENT, the relationship that exists is that employees are assigned to departments.

After you define the entities, their primary keys, and their relationships, you can define additional attributes for the entities. In the case of the EMPLOYEE entity, you might define the following additional attributes:

- Birth date
- Hire date
- Home address
- Office phone number
- Gender
- Resume

You can read more about defining attributes later in this information.

Finally, you normalize the data.

Related concepts

Referential constraints
Db2 ensures referential integrity between your tables when you define referential constraints.

Db2 keys
A *key* is a column or an ordered collection of columns that is identified in the description of a table, an index, or a referential constraint. Keys are crucial to the table structure in a relational database.

Normalization in database design
Normalization helps you avoid redundancies and inconsistencies in your data. There are several forms of normalization.

Entities for different types of relationships
In a relational database, separate entities must be defined for different types of relationships.

Entities for different types of relationships

In a relational database, separate entities must be defined for different types of relationships.

In a relational database, you can express several types of relationships. Consider the possible relationships between employees and departments. A given employee can work in only one department; this relationship is *one-to-one* for employees. One department usually has many employees; this relationship is *one-to-many* for departments. Relationships can be one-to-many, many-to-one, one-to-one, or many-to-many.

The type of a given relationship can vary, depending on the specific environment. If employees of a company belong to several departments, the relationship between employees and departments is many-to-many.

You need to define separate entities for different types of relationships. When modeling relationships, you can use diagram conventions to depict relationships by using different styles of lines to connect the entities.

One-to-one relationships

A *one-to-one relationship* in database design is bidirectional relationship, which means it is single-valued in both directions.

For example, an employee has a single resume; each resume belongs to only one person. The following figure illustrates that a one-to-one relationship exists between the two entities. In this case, the relationship reflects the rules that an employee can have only one resume and that a resume can belong to only one employee.

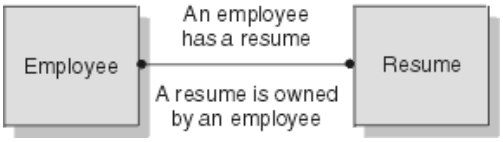


Figure 56. Assigning one-to-one facts to an entity

One-to-many relationships

A *one-to-many relationship* occurs in database design when one entity has a multivalued relationship with another entity.

In the following figure, you see that a one-to-many relationship exists between two entities—employee and department. This figure reinforces the business rules that a department can have many employees, but that each individual employee can work for only one department.

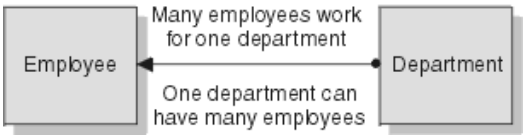


Figure 57. Assigning many-to-one facts to an entity

Many-to-many relationships

A *many-to-many relationship* is a relationship that is multivalued in both directions.

The following figure illustrates this kind of relationship. An employee can work on more than one project, and a project can have more than one employee assigned.

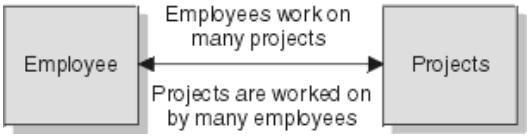


Figure 58. Assigning many-to-many facts to an entity

If you look at this information's example tables, you can find answers for the following questions:

- What does Wing Lee work on?
- Who works on project number OP2012?

Both questions yield multiple answers. Wing Lee works on project numbers OP2011 and OP2012. The employees who work on project number OP2012 are Ramlal Mehta and Wing Lee.

Application of business rules to relationships

Whether a given relationship is one-to-one, one-to-many, many-to-one, or many-to-many, your relationships need to make good business sense.

Database designers and data analysts can be more effective when they have a good understanding of the business. If they understand the data, the applications, and the business rules, they can succeed in building a sound database design.

When you define relationships, you have a large influence on how smoothly your business runs. If you perform this task poorly, your database and associated applications are likely to have many problems, some of which might not manifest themselves for years.

Related concepts

Application of business rules to relationships
Whether a given relationship is one-to-one, one-to-many, many-to-one, or many-to-many, your relationships need to make good business sense.

Creation of relationships with referential constraints
Referential integrity is a condition in which all intended references from data in one table column to data in another table column are valid. By using referential constraints, you can define relationships between entities that you define in Db2.

Entities for different types of relationships (Db2 Administration Guide)

Related tasks

Creating tables for data integrity (Db2 Application programming and SQL)

Altering a table for referential integrity (Db2 Administration Guide)

Entity attributes in database design

When you define attributes for entities, you generally work with the data administrator to decide on names, data types, and appropriate values for the attributes.

Naming conventions for entity attributes

Naming conventions for attributes help database designers ensure consistency within an organization. Most organizations have naming conventions. In addition to following these conventions, data administrators also base attribute definitions on class words. A *class word* is a single word that indicates the nature of the data that the attribute represents.

Example: The class word NUMBER indicates an attribute that identifies the number of an entity. Attribute names that identify the numbers of entities should therefore include the class word of NUMBER. Some examples are EMPLOYEE_NUMBER, PROJECT_NUMBER, and DEPARTMENT_NUMBER.

When an organization does not have well-defined guidelines for attribute names, the data administrators try to determine how the database designers have historically named attributes. Problems occur when multiple individuals are inventing their own naming schemes without consulting each other.

Data types for entity attributes

A data type must be specified for each attribute. Most organizations have well-defined guidelines for using the different data types. Here is an overview of the main data types that you can use for the attributes of your entities.

String data types

Data that contains a combination of letters, numbers, and special characters. String data types are listed below:

- CHARACTER: Fixed-length character strings. The common short name for this data type is CHAR.
- VARCHAR: Varying-length character strings.
- CLOB: Varying-length character large object strings, typically used when a character string might exceed the limits of the VARCHAR data type.
- GRAPHIC: Fixed-length graphic strings that contain double-byte characters.

- VARGRAPHIC: Varying-length graphic strings that contain double-byte characters.
- DBCLOB: Varying-length strings of double-byte characters in a large object.
- BINARY: A sequence of bytes that is not associated with a code page.
- VARBINARY: Varying-length binary strings.
- BLOB: Varying-length binary strings in a large object.
- XML: Varying-length string that is an internal representation of XML.

Numeric data types

Data that contains digits. Numeric data types are listed below:

- SMALLINT: for small integers.
- INTEGER: for large integers.
- BIGINT: for bigger values.
- DECIMAL(*p,s*) or NUMERIC(*p,s*), where *p* is precision and *s* is scale: for packed decimal numbers with precision *p* and scale *s*. *Precision* is the total number of digits, and *scale* is the number of digits to the right of the decimal point.
- DECFLOAT: for decimal floating-point numbers.
- REAL: for single-precision floating-point numbers.
- DOUBLE: for double-precision floating-point numbers.

Datetime data types

Data values that represent dates, times, or timestamps. Datetime data types are listed below:

- DATE: Dates with a three-part value that represents a year, month, and day.
- TIME: Times with a three-part value that represents a time of day in hours, minutes, and seconds.
- TIMESTAMP: Timestamps with a seven-part value that represents a date and time by year, month, day, hour, minute, second, and microsecond.

Examples: You might use the following data types for attributes of the EMPLOYEE entity:

- EMPLOYEE_NUMBER: CHAR(6)
- EMPLOYEE_LAST_NAME: VARCHAR(15)
- EMPLOYEE_HIRE_DATE: DATE
- EMPLOYEE_SALARY_AMOUNT: DECIMAL(9,2)

The data types that you choose are business definitions of the data type. During physical database design you might need to change data type definitions or use a subset of these data types. The database or the host language might not support all of these definitions, or you might make a different choice for performance reasons.

For example, you might need to represent monetary amounts, but Db2 and many host languages do not have a data type MONEY. In the United States, a natural choice for the SQL data type in this situation is DECIMAL(10,2) to represent dollars. But you might also consider the INTEGER data type for fast, efficient performance.

Values for key attributes

When you design a database, you must decide what values are acceptable for the various attributes of an entity. For example, you would not want to allow numeric data in an attribute for a person's name. The data types that you choose limit the values that apply to a given attribute, but you can also use other mechanisms. These other mechanisms are domains, null values, and default values.

Domains

A *domain* describes the conditions that an attribute value must meet to be a valid value. Sometimes the domain identifies a range of valid values. By defining the domain for a particular attribute, you apply business rules to ensure that the data makes sense.

Examples:

- A domain might state that a phone number attribute must be a 10-digit value that contains only numbers. You would not want the phone number to be incomplete, nor would you want it to contain alphabetic or special characters and thereby be invalid. You could choose to use either a numeric data type or a character data type. However, the domain states the business rule that the value must be a 10-digit value that consists of numbers. Before finalizing this rule, consider if you have a need for international phone numbers, which have different formats.
- A domain might state that a month attribute must be a 2-digit value from 01 to 12. Again, you could choose to use datetime, character, or numeric data types for this value, but the domain demands that the value must be in the range of 01 through 12. In this case, incorporating the month into a datetime data type is probably the best choice. This decision should be reviewed again during physical database design.

Null values

A *null value* is a special indicator that represents the absence of a value. When you are designing attributes for your entities, you will sometimes find that an attribute does not have a value for every instance of the entity. For example, you might want an attribute for a person's middle name, but you can't require a value because some people have no middle name. For these occasions, you can define the attribute so that it can contain null values.

The value can be absent because it is unknown, not yet supplied, or nonexistent. The DBMS treats the null value as an actual value, not as a zero value, a blank, or an empty string.

Just as some attributes should be allowed to contain null values, other attributes should not contain null values.

Example: For the EMPLOYEE entity, you might not want to allow the attribute EMPLOYEE_LAST_NAME to contain a null value.

Default values

A *default value* is a value that applies to an attribute if no other valid value is available. In some cases, you might not want a specific attribute to contain a null value, but you don't want to require that the user or program always provide a value. In this case, a default value might be appropriate.

Example: Assume that you don't want the EMPLOYEE_HIRE_DATE attribute to contain null values and that you don't want to require users to provide this data. If data about new employees is generally added to the database on the employee's first day of employment, you could define a default value of the current date.

Related concepts

Entities for different types of relationships
In a relational database, separate entities must be defined for different types of relationships.

Normalization in database design

Normalization helps you avoid redundancies and inconsistencies in your data. There are several forms of normalization.

After you define entities and decide on attributes for the entities, you normalize entities to avoid redundancy. An entity is normalized if it meets a set of constraints for a particular normal form, which this information describes. Entities can be in first, second, third, and fourth normal forms, each of which has certain rules that are associated with it. In some cases, you follow these rules, and in other cases, you do not follow them.

The rules for normal form are cumulative. In other words, for an entity to satisfy the rules of second normal form, it also must satisfy the rules of first normal form. An entity that satisfies the rules of fourth normal form also satisfies the rules of first, second, and third normal form.

In the context of logical data modeling, an *instance* is one particular occurrence. An instance of an entity is a set of data values for all the attributes that correspond to that entity.

Example: The following figure shows one instance of the EMPLOYEE entity.

EMPLOYEE

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME	DEPARTMENT _NUMBER	EMPLOYEE _HIRE _DATE	JOB _NAME	EDUCATION _LEVEL	EMPLOYEE _YEARLY _SALARY _AMOUNT	COMMISSION _AMOUNT
000010	CHRISTINE	HAAS	A00	1965-01-01	PRES	18	52750.00	4220.00

Figure 59. One instance of an entity

First normal form

A relational entity satisfies the requirement of *first normal form* if every instance of the entity contains only one value, but never multiple repeating attributes.

Repeating attributes, often called a *repeating group*, are different attributes that are inherently the same. In an entity that satisfies the requirement of first normal form, each attribute is independent and unique in its meaning and its name. For example, assume that an entity contains the following attributes:

```
EMPLOYEE_NUMBER
JANUARY_SALARY_AMOUNT
FEBRUARY_SALARY_AMOUNT
MARCH_SALARY_AMOUNT
```

This situation violates the requirement of first normal form, because JANUARY_SALARY_AMOUNT, FEBRUARY_SALARY_AMOUNT, and MARCH_SALARY_AMOUNT are essentially the same attribute, EMPLOYEE_MONTHLY_SALARY_AMOUNT.

Second normal form

An entity is in *second normal form* if each attribute that is not in the primary key provides a fact that depends on the entire key. A violation of the second normal form occurs when a non-primary key attribute is a fact about a subset of a composite key. For example, an inventory entity records quantities of specific parts that are stored at particular warehouses. The following figure shows the attributes of the inventory entity.



Figure 60. A primary key that violates second normal form

Example:

Here, the primary key consists of the PART and the WAREHOUSE attributes together. Because the attribute WAREHOUSE_ADDRESS depends only on the value of WAREHOUSE, the entity violates the rule for second normal form. This design causes several problems:

- Each instance for a part that this warehouse stores repeats the address of the warehouse.
- If the address of the warehouse changes, every instance referring to a part that is stored in that warehouse must be updated.
- Because of the redundancy, the data might become inconsistent. Different instances could show different addresses for the same warehouse.
- If at any time the warehouse has no stored parts, the address of the warehouse might not exist in any instances in the entity.

To satisfy second normal form, the information in the figure above would be in two entities, as the following figure shows.



Figure 61. Two entities that satisfy second normal form

Third normal form

An entity is in *third normal form* if each non-primary key attribute provides a fact that is independent of other non-key attributes and depends only on the key.

A violation of the third normal form occurs when a non-primary attribute is a fact about another non-key attribute. For example, the first entity in the following figure contains the attributes EMPLOYEE_NUMBER and DEPARTMENT_NUMBER. Suppose that a program or user adds an attribute, DEPARTMENT_NAME, to the entity. The new attribute depends on DEPARTMENT_NUMBER, whereas the primary key is on the EMPLOYEE_NUMBER attribute. The entity now violates third normal form.

Changing the DEPARTMENT_NAME value based on the update of a single employee, David Brown, does not change the DEPARTMENT_NAME value for other employees in that department. The updated version of the entity in the following figure illustrates the resulting inconsistency. Additionally, updating the DEPARTMENT_NAME in this table does not update it in any other table that might contain a DEPARTMENT_NAME column.

Employee_Department table before update

Key

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME	DEPARTMENT _NUMBER	DEPARTMENT _NAME
000200	DAVID	BROWN	D11	MANUFACTURING SYSTEMS
000320	RAMLAL	MEHTA	E21	SOFTWARE SUPPORT
000220	JENNIFER	LUTZ	D11	MANUFACTURING SYSTEMS

Employee_Department table after update

Key

EMPLOYEE _NUMBER	EMPLOYEE _FIRST _NAME	EMPLOYEE _LAST _NAME	DEPARTMENT _NUMBER	DEPARTMENT _NAME
000200	DAVID	BROWN	D11	INSTALLATION MGMT
000320	RAMLAL	MEHTA	E21	SOFTWARE SUPPORT
000220	JENNIFER	LUTZ	D11	MANUFACTURING SYSTEMS

Figure 62. The update of an unnormalized entity

You can normalize the entity by modifying the EMPLOYEE_DEPARTMENT entity and creating two new entities: EMPLOYEE and DEPARTMENT. The following figure shows the new entities. The DEPARTMENT entity contains attributes for DEPARTMENT_NUMBER and DEPARTMENT_NAME. Now, an update such as changing a department name is much easier. You need to make the update only to the DEPARTMENT entity.

Employee table

Key

EMPLOYEE_NUMBER	EMPLOYEE_FIRST_NAME	EMPLOYEE_LAST_NAME
000200	DAVID	BROWN
000329	RAMLAL	MEHTA
000220	JENNIFER	LUTZ

Department table

Key

DEPARTMENT_NUMBER	DEPARTMENT_NAME
D11	MANUFACTURING SYSTEMS
E21	SOFTWARE SUPPORT

Employee_Department table

Key

DEPARTMENT_NUMBER	EMPLOYEE_NUMBER
D11	000200
D11	000220
E21	000329

Figure 63. Normalized entities: EMPLOYEE, DEPARTMENT, and EMPLOYEE_DEPARTMENT

Fourth normal form

An entity is in fourth normal form if no instance contains two or more independent, multivalued facts about an entity.

For example, Consider the EMPLOYEE entity. Each instance of EMPLOYEE could have both SKILL_CODE and LANGUAGE_CODE. An employee can have several skills and know several languages. Two relationships exist, one between employees and skills, and one between employees and languages. An entity is not in fourth normal form if it represents both relationships, as the following figure shows.



Figure 64. An entity that violates fourth normal form

Instead, you can avoid this violation by creating two entities that represent both relationships, as the following figure shows.



Figure 65. Entities that are in fourth normal form

If, however, the facts are interdependent (that is, the employee applies certain languages only to certain skills) you should not split the entity.

You can put any data into fourth normal form. A good rule to follow when doing logical database design is to arrange all the data in entities that are in fourth normal form. Then decide whether the result gives you an acceptable level of performance. If the performance is not acceptable, denormalizing your design is a good approach to improving performance.

Related concepts

Database design with denormalization

The rules of normalization do not consider performance. In some cases, you need to consider denormalization to improve performance.

Logical database design with Unified Modeling Language

UML modeling is based on object-oriented programming principals. UML defines a standard set of modeling diagrams for all stages of developing a software system.

This information describes the entity-relationship model of database design. Another model that you can use is Unified Modeling Language (UML). The Object Management Group is a consortium that created the UML standard. This topic provides a brief overview of UML.

The basic difference between the entity-relationship model and the UML model is that, instead of designing entities as this information illustrates, you model objects. Conceptually, UML diagrams are like the blueprints for the design of a software development project.

Some examples of UML diagrams are listed below:

Class

Identify high-level entities, known as classes. A *class* describes a set of objects that have the same attributes. A class diagram shows the relationships between classes.

Use case

Presents a high-level view of a system from the user's perspective. A *use case* diagram defines the interactions between users and applications or between applications. These diagrams graphically depict system behavior. You can work with use-case diagrams to capture system requirements, learn how the system works, and specify system behavior.

Activity
Models the workflow of a business process, typically by defining rules for the sequence of activities in the process. For example, an accounting company can use activity diagrams to model financial transactions.

Interaction
Shows the required sequence of interactions between objects. Interaction diagrams can include sequence diagrams and collaboration diagrams.

- Sequence diagrams show object interactions in a time-based sequence that establishes the roles of objects and helps determine class responsibilities and interfaces.
- Collaboration diagrams show associations between objects that define the sequence of messages that implement an operation or a transaction.

Component
Shows the dependency relationships between components, such as main programs, and subprograms.

Many available tools from the WebSphere and Rational product families ease the task of creating a UML model. Developers can graphically represent the architecture of a database and how it interacts with applications using the following UML modeling tools:

- WebSphere Business Integration Workbench, which provides a UML modeler for creating standard UML diagrams.
- A WebSphere Studio Application Developer plug-in for modeling Java and web services applications and for mapping the UML model to the entity-relationship model.
- Rational Rose® Data Modeler, which provides a modeling environment that connects database designers who use entity-relationship modeling with developers of OO applications.
- Rational Rapid Developer, an end-to-end modeler and code generator that provides an environment for rapid design, integration, construction, and deployment of web, wireless, and portal-based business applications.
- IBM Rational Data Architect (RDA) has rich functionality that gives data professionals the ability to design a relational or federated database, and perform impact analysis across models.

Similarities exist between components of the entity-relationship model and UML diagrams. For example, the class structure corresponds closely to the entity structure.

Using the modeling tool Rational Rose Data Modeler, developers use a specific type of diagram for each type of development model:

- Business models—Use case diagram, activity diagram, sequence diagram
- Logical data models or application models—Class diagram
- Physical data models—Data model diagram

The logical data model provides an overall view of the captured business requirements as they pertain to data entities. The data model diagram graphically represents the physical data model. The physical data model uses the logical data model's captured requirements, and applies them to specific DBMS languages. Physical data models also capture the lower-level detail of a DBMS database.

Database designers can customize the data model diagram from other UML diagrams, which enables them to work with concepts and terminology, such as columns, tables, and relationships, with which they are already familiar. Developers can also transform a logical data model into to a physical data model.

Because the data model diagram includes diagrams for modeling an entire system, it enables database designers, application developers, and other development team members to share and track business requirements throughout the development process. For example, database designers can capture information, such as constraints, triggers, and indexes directly on the UML diagram. Developers can also transfer between object and data models and use basic transformation types such as many-to-many relationships.

Related concepts

Logical database design using entity-relationship modeling
Before you implement a database, you should plan or design it so that it satisfies all requirements. This first task of designing a database is called logical design.

Physical database design
The physical design of your database optimizes performance while ensuring data integrity by avoiding unnecessary data redundancies. During physical design, you transform the entities into tables, the instances into rows, and the attributes into columns.

Physical database design

The physical design of your database optimizes performance while ensuring data integrity by avoiding unnecessary data redundancies. During physical design, you transform the entities into tables, the instances into rows, and the attributes into columns.

After completing the logical design of your database, you now move to the physical design. You and your colleagues need to make many decisions that affect the physical design, some of which are listed below.

- How to translate entities into physical tables
- What attributes to use for columns of the physical tables
- Which columns of the tables to define as keys
- What indexes to define on the tables
- What views to define on the tables
- How to denormalize the tables
- How to resolve many-to-many relationships
- What designs can take advantage of hash access

Physical design is the time when you abbreviate the names that you chose during logical design. For example, you can abbreviate the column name that identifies employees, EMPLOYEE_NUMBER, to EMPNO. In Db2 for z/OS, you need to abbreviate column names and table names to fit the physical constraint of a 30-byte maximum for column names and a 128-byte maximum for table names.

The task of building the physical design is a job that truly never ends. You need to continually monitor the performance and data integrity characteristics of the database as time passes. Many factors necessitate periodic refinements to the physical design.

Db2 lets you change many of the key attributes of your design with ALTER SQL statements. For example, assume that you design a partitioned table so that it stores 36 months' worth of data. Later you discover that you need to extend that design to 84 months' worth of data. You can add or rotate partitions for the current 36 months to accommodate the new design.

The remainder of this information includes some valuable information that can help you as you build and refine the physical design of your database. However, this task generally requires more experience with Db2 than most readers of this introductory level information are likely to have.

Related concepts

Logical database design with Unified Modeling Language
UML modeling is based on object-oriented programming principals. UML defines a standard set of modeling diagrams for all stages of developing a software system.

Logical database design using entity-relationship modeling
Before you implement a database, you should plan or design it so that it satisfies all requirements. This first task of designing a database is called logical design.

Database design with denormalization

The rules of normalization do not consider performance. In some cases, you need to consider denormalization to improve performance.

During physical design, analysts transform the entities into tables and the attributes into columns. Consider the example in again. The warehouse address column first appears as part of a table that

contains information about parts and warehouses. To further normalize the design of the table, analysts remove the warehouse address column from that table. Analysts also define the column as part of a table that contains information only about warehouses.

Normalizing tables is generally the recommended approach. What if applications require information about both parts and warehouses, including the addresses of warehouses? The premise of the normalization rules is that SQL statements can retrieve the information by joining the two tables. The problem is that, in some cases, performance problems can occur as a result of normalization. For example, some user queries might view data that is in two or more related tables; the result is too many joins. As the number of tables increases, the access costs can increase, depending on the size of the tables, the available indexes, and so on. For example, if indexes are not available, the join of many large tables might take too much time. You might need to denormalize your tables. *Denormalization* is the intentional duplication of columns in multiple tables, and it increases data redundancy.

Example 1: Consider the design in which both tables have a column that contains the addresses of warehouses. If this design makes join operations unnecessary, it could be a worthwhile redundancy. Addresses of warehouses do not change often, and if one does change, you can use SQL to update all instances fairly easily.

Tip: Do not automatically assume that all joins take too much time. If you join normalized tables, you do not need to keep the same data values synchronized in multiple tables. In many cases, joins are the most efficient access method, despite the overhead they require. For example, some applications achieve 44-way joins in subsecond response time.

When you build your physical design, you and your colleagues need to decide whether to denormalize the data. Specifically, you need to decide whether to combine tables or parts of tables that are frequently accessed by joins that have high-performance requirements. This is a complex decision about which this information cannot give specific advice. To make the decision, you need to assess the performance requirements, different methods of accessing the data, and the costs of denormalizing the data. You need to consider the trade-off; is duplication, in several tables, of often-requested columns less expensive than the time for performing joins?

Recommendations:

- Do not denormalize tables unless you have a good understanding of the data and the business transactions that access the data. Consult with application developers before denormalizing tables to improve the performance of users' queries.
- When you decide whether to denormalize a table, consider all programs that regularly access the table, both for reading and for updating. If programs frequently update a table, denormalizing the table affects performance of update programs because updates apply to multiple tables rather than to one table.

In the following figure, information about parts, warehouses, and warehouse addresses appear in two tables, both in normal form.



Figure 66. Two tables that satisfy second normal form

The following figure illustrates the denormalized table.



Figure 67. Denormalized table

Resolving many-to-many relationships is a particularly important activity because doing so helps maintain clarity and integrity in your physical database design. To resolve many-to-many relationships, you introduce *associative tables*, which are intermediate tables that you use to tie, or associate, two tables to each other.

Example 2: Employees work on many projects. Projects have many employees. In the logical database design, you show this relationship as a many-to-many relationship between project and employee. To resolve this relationship, you create a new associative table, EMPLOYEE_PROJECT. For each combination of employee and project, the EMPLOYEE_PROJECT table contains a corresponding row. The primary key for the table would consist of the employee number (EMPNO) and the project number (PROJNO).

Another decision that you must make relates to the use of repeating groups.

Example 3: Assume that a heavily used transaction requires the number of wires that are sold by month in a specific year. Performance factors might justify changing a table so that it violates the rule of first normal form by storing repeating groups. In this case, the repeating group would be: MONTH, WIRE. The table would contain a row for the number of sold wires for each month (January wires, February wires, March wires, and so on).

Recommendation: If you decide to denormalize your data, document your denormalization thoroughly. Describe, in detail, the logic behind the denormalization and the steps that you took. Then, if your organization ever needs to normalize the data in the future, an accurate record is available for those who must do the work.

Related concepts

Creation of indexes
Indexes provide efficient access to table data, but can require additional processing when you modify data in a table.

Db2 performance management
Managing the performance of a Db2 subsystem involves understanding a wide range of system components. You need to understand the performance of those components, how to monitor the components, and how to identify problem areas.

Entity normalization (Db2 Administration Guide)

Normalization in database design
Normalization helps you avoid redundancies and inconsistencies in your data. There are several forms of normalization.

Customized data views

A view offers an alternative way of describing data that exists in one or more tables.

Some users might find that no single table contains all the data that they need; rather, the data might be scattered among several tables. Furthermore, one table might contain more data than users want to see or more than you want to authorize them to see. For those situations, you can create views.

You might want to use views for a variety of reasons:

- To limit access to certain kinds of data

  You can create a view that contains only selected columns and rows from one or more tables. Users with the appropriate authorization on the view see only the information that you specify in the view definition.

  Example: You can define a view on the EMP table to show all columns except for SALARY and COMM (commission). You can grant access to this view to people who are not managers because you probably don't want them to have access to this kind of information.

- To combine data from multiple tables

  You can create a view that uses one of the set operators, UNION, INTERSECT, or EXCEPT, to logically combine data from intermediate result tables. Additionally, you can specify either DISTINCT (the default) or ALL with a set operator. You can query a view that is defined with a set operator as if it were one large result table.

  Example: Assume that three tables contain data for a time period of one month. You can create a view that is the UNION ALL of three fullselects, one for each month of the first quarter of 2004. At the end of the third month, you can view comprehensive quarterly data.

You can create a view any time after the underlying tables exist. The owner of a set of tables implicitly has the authority to create a view on them. A user with administrative authority at the system or database level can create a view for any owner on any set of tables. If they have the necessary authority, other users can also create views on a table that they didn't create.

Related concepts

Authorization and security mechanisms for data access
Authorization is an important part of controlling Db2. The security and authorization mechanisms that control access to Db2 data are both direct and indirect.

A view that combines information from several tables
You can create a view that contains a union of more than one table. A union of more than one table is called a *join*.

Database design with indexes

You can use indexes to optimize data access, to ensure uniqueness, and to enable clustering.

If you are involved in the physical design of a database, you work with other designers to determine what columns and expressions you should index. You use process models that describe how different applications are going to access the data. This information is very important when you decide on indexing strategies to ensure adequate performance.

The main purposes of an index are:

To optimize data access
In many cases, access to data is faster with an index than without an index. If the DBMS uses an index to find a row in a table, the scan can be faster than when the DBMS scans an entire table.

To ensure uniqueness
A table with a unique index cannot have two rows with the same values in the column or columns that form the index key.

Example: If payroll applications use employee numbers, no two employees can have the same employee number.

To enable clustering
A clustering index keeps table rows in a specified sequence to minimize page access for a set of rows. When a table space is partitioned, rows are clustered within each partition. Clustering can be in the same order as the partitioning.

Example: If the partition is on the month and the clustering index is on the name, the rows are clustered on the name within the month.

In general, users of the table are unaware that an index is in use. Db2 decides whether to use the index to access the table.

Related concepts

Creation of indexes
Indexes provide efficient access to table data, but can require additional processing when you modify data in a table.

Related tasks

Designing indexes for performance (Db2 Performance)

Related information

Implementing Db2 indexes (Db2 Administration Guide)

Database design with hash access

You can organize tables for hash access to individual data rows. Hash-organized table spaces are deprecated and likely to be unsupported in the future.

Deprecated function:

Hash-organized tables are deprecated. Beginning in Db2 12, packages that are bound with APPLCOMPAT(V12R1M504) or higher cannot create hash-organized tables or alter existing tables to use

hash-organization. Existing hash-organized tables remain supported, but they are likely to be unsupported in the future.

If you are involved in the physical design of a database, you work with other designers to determine when to enable hash access on tables.

The main purposes of hash organization is for programs that regularly access a single row in a table and the table has a unique identifier for each row. You can use hash access to directly retrieve the data from individual rows without scanning the index or the table space for the matching equal predicate. Hash access is faster and more efficient than table scans and index scans, but tables that have hash access enabled require more disk space. Hash access requires that tables have at least one column with values that are unique to each row.

Related concepts

Db2 hash spaces (deprecated)
A *hash space* is a defined disk space that organizes table data for hash access. Hash-organized table spaces are deprecated and likely to be unsupported in the future.

Hash access paths
Db2 can use hash access paths to directly access a single row in a table and avoid scanning the table or index.

Hash access (ACCESSTYPE='H', 'HN', or 'MH') (Db2 Performance)

Related tasks

Organizing tables for hash access to individual rows (deprecated) (Db2 Performance)
Creating tables that use hash organization (deprecated) (Db2 Administration Guide)
Altering tables for hash access (deprecated) (Db2 Administration Guide)
Monitoring hash access (deprecated) (Db2 Performance)
Managing space and page size for hash-organized tables (Db2 Performance)

Row design for effective space use

Record size is an important consideration in the design of a table. In Db2, a *record* is the storage representation of a row.

Db2 stores records within pages that are 4 KB, 8 KB, 16 KB, or 32 KB in size. Generally, you cannot create a table with a maximum record size that is greater than the page size. No other absolute limit exists, but you risk wasting storage space if you ignore record size in favor of implementing a good theoretical design.

If the record length is larger than the page size, increase the page size or consider using a large object (LOB) data type or an XML data type.

Record lengths and page size

The sum of the lengths of all the columns is the *record length*. The length of data that is physically stored in the table is the record length plus Db2 overhead for each row and each page. You can choose various page sizes for record lengths that best fit your needs.

If row sizes are very small, use the 4 KB page size. Use the default of 4-KB page sizes when access to your data is random and typically requires only a few rows from each page.

Some situations require larger page sizes. Db2 provides three larger page sizes of 8 KB, 16 KB, and 32 KB to allow for longer records. For example, when the size of individual rows is greater than 4-KB, you must use a larger page size. In general, you can improve performance by using pages for record lengths that best suit your needs.

Avoiding wasted space

If a table space contains large records that use up most of the page size and cannot fit additional records, that database design wastes space. In general, space is wasted in a table space that contains only records that are slightly longer than half a page because a page can hold only one record. If you can

reduce the record length to just under half a page, you need only half as many pages. Similar considerations apply to records that are just over a third of a page, a quarter of a page, and so on. In these situations, you can use compression or increase the page size.

Related concepts

Large object data types
You can use large object data types to store audio, video, images, and other files that are larger than 32 KB.

pureXML
You can use pureXML with your client applications to manage XML data in Db2 tables. You can store well-formed XML documents in their hierarchical form and retrieve all or portions of those documents.

XML and Db2

You can use XML in a Db2 database. *XML*, which stands for Extensible Markup Language, is a text-based tag language. Its style is similar to HTML, except that XML users can define their own tags.

The explosive growth of the Internet was a catalyst for the development and industry-wide acceptance of XML. Because of the dramatic increase of on demand business applications, organizations need to exchange data in a robust, open format. The options that were available before the development of XML were Standard Generalized Markup Language (SGML) and HTML. SGML is too complex for wide use on the web. HTML is good for the presentation of web pages, but it is not designed for the easy exchange of information. XML has emerged as a useful and flexible simplification of SGML that enables the definition and processing of documents for exchanging data between businesses, between applications, and between systems.

You can think of HTML as a way of communicating information between computers and people. You can think of XML as a way of communicating information between computers. You can convert XML to HTML so that people can view the information.

Benefits of using XML with Db2 for z/OS

With XML data in Db2 for z/OS, organizations can gain the following benefits:

Improved customer relationships
XML lets you deliver personalized information to customers, enable new distribution channels, and respond faster to customer needs.

Optimized internal operations
With XML, you can drive business data from your existing systems to the Web. XML enables you to automate transactions that do not require human interaction.

Maximized partnerships
Because of the widespread use of XML in the industry, you can easily share information with suppliers, buyers, and partners.

Tools and software
You can take advantage of many XML tools and software, such as WebSphere Studio, XML parsers and processors, and the SQL/XML publishing function.

XML vocabularies exist for specific industries to help organizations in those industries standardize their use of XML. An *XML vocabulary* is an XML description that is designed for a specific purpose. Widespread industry use of XML has resulted in more effective and efficient business-to-business transactions.

Ways to use XML with Db2 11 for z/OS

Organizations use XML for document processing and for publishing information on the web. There are various publishing functions and tools that help you integrate XML with Db2 data.

To integrate XML with Db2 data, you can use the SQL/XML publishing functions. The native XML, or pureXML, support in Db2 offers efficient and versatile capabilities for managing your XML data. Db2 stores and processes XML in its inherent hierarchical format, avoiding the performance and flexibility limitations that occur when XML is stored as text in CLOBs or mapped to relational tables.

SQL/XML publishing functions allow applications to generate XML data from relational data. XML is a popular choice when you want to send Db2 data to another system or to another application in a common format. You can choose from one of several ways to publish XML documents:

- Use SQL/XML functions that are integrated with Db2.

  Db2 integrates SQL/XML publishing functions into the Db2 product. A set of SQL built-in functions allows applications to generate XML data from Db2 data with high performance. The SQL/XML publishing functions can reduce application development efforts in generating XML for data integration, information exchange, and web services.
- Take advantage of Db2 web services support.

  Web services provide a way for programs to invoke other programs, typically on the Internet, that transmit input parameters and generate results as XML.
- Use a tool to code XML.

  WebSphere Studio provides a development environment for publishing XML documents from relational data.

Chapter 7. Db2 for z/OS architecture

z/OS and the IBM System z10 and subsequent 64-bit z/Architecture processors offer architecture that provides qualities of service that are critical for e-business.

Related concepts

Db2 data servers and environments

Db2 data server products run on a wide set of operating systems, including z/OS, IBM i, Linux, UNIX, and Windows.

z/Architecture and the z/OS operating system

z/OS, which is highly secure, scalable, and open, offers high-performance that supports a diverse application execution environment. The tight integration that Db2 has with the IBM Z architecture and the z/OS environment creates a synergy that allows Db2 to exploit advanced z/OS functions.

The z/OS operating system is based on 64-bit z/Architecture. Db2 11 for z/OS operates on IBM System z10 and subsequent 64-bit z/Architecture processors.

Db2 gains a tremendous benefit from z/Architecture. The architecture of Db2 for z/OS takes advantage of the key z/Architecture benefit: 64-bit virtual addressing support. With 64-bit z/Architecture, Db2 gains an immediate scalability benefit.

The following z/Architecture features benefit Db2:

64-bit storage
Increased capacity of central memory from 2 GB to 16 exabytes eliminates most storage constraints. 64-bit storage also allows for 16 exabytes of virtual address space, a huge step in the continuing evolution of increased virtual storage. In addition to improving Db2 performance, 64-bit storage improves availability and scalability, and it simplifies storage management.

High-speed communication
HiperSockets enable high-speed TCP/IP communication across partitions of the same IBM Z server, for example, between Linux on IBM Z and Db2 for z/OS.

Dynamic workload management
The z/OS Workload Manager (WLM) provides solutions for managing workload distribution, workload balancing, and distributing resources to competing workloads. z/OS workload management is the combined cooperation of various subsystems (CICS, IMS/ESA®, JES, APPC, TSO/E, z/OS UNIX System Services, DDF, Db2, LSFM, and Internet Connection Server) with z/OS workload management. The Intelligent Resource Director (IRD) allows you to group logical partitions that are resident on the same physical server, and in the same sysplex, into an LPAR cluster. This gives Workload Manager the ability to manage resources across the entire cluster of logical partitions.

Specialty engines
With special processors, such as the IBM Z Integrated Information Processor (zIIP), Db2 achieves higher degrees of query parallelism and higher levels of transaction throughput. The zIIP is designed to improve resource optimization and lower the cost of eligible workloads, enhancing the role of the mainframe as the data hub of the enterprise.

In addition to the benefits of z/Architecture, Db2 takes advantage of many other features of the z/OS operating system:

High security
z/OS and its predecessors have provided robust security for decades. Security features deliver privacy for users, applications, and data, and these features protect the integrity and isolation of running processes. Current security functions have evolved to include comprehensive network and transaction security that operates with many other operating systems. Enhancements to the z/OS Security Server provide improved security options, such as multilevel security. The IBM Z environment offers highly secure cryptographic functions and provides improved Secure Sockets Layer (SSL) performance.

© Copyright IBM Corp. 2001, 2020

Open software technologies
z/OS supports the latest open software technologies that include Enterprise JavaBeans, XML, and Unicode.

Cluster technology
The z/OS Parallel Sysplex provides cluster technology that achieves availability 24 hours a day, 7 days a week. Cluster technology also provides the capability for horizontal growth. Horizontal growth solves the problems of performance overheads and system management issues that you typically encounter when combining multiple machines to access the same database. With horizontal growth, you achieve more scalability; your system can grow beyond the confines of a single machine while your database remains intact.

Solid-state drives
Solid-state drives (SSDs) are more reliable, consume less power, and generate less heat than traditional hard disk drives (HDDs). SSDs can also improve the performance of online transaction processing. SSDs are especially efficient at performing random access requests, and they provide greater throughput than HDDs. Some IBM System Storage® series allow a combination of HDDs and SSDs.

Parallel Access Volume (PAV)
IBM Enterprise Storage Server® (ESS) exploits the Parallel Access Volume and Multiple Allegiance features of z/OS and supports up to 256 I/Os per logical disk volume. A single z/OS host can issue I/Os in parallel to the same logical volume, and different hosts can issue I/Os to a shared volume in parallel.

HyperPAV
HyperPAV is available on some IBM System Storage series. HyperPAV helps applications to achieve equal or greater I/O performance than the original PAV feature, but uses fewer z/OS resources.

Adaptive multi-stream prefetching
Adaptive multi-stream prefetching (AMP) is a sequential prefetching algorithm that resolves cache pollution and prefetch waste for a cache that is shared by multiple sequential request streams. AMP works well to manage caches efficiently across a wide variety of workloads and cache sizes.

Cache optimization
Db2 code and control structures are adapted to reduce cache misses.

MIDAW
The IBM Z environment also supports the Modified Indirect Data Address Word (MIDAW) facility, which is designed to improve channel utilization and throughput, and which can potentially reduce I/O response times.

FICON® channels
These channels offer significant performance benefits for transaction workloads. FICON features, such as a rapid data transfer rate (4 GB per second), also result in faster table scans and improved utility performance.

High performance FICON
High Performance FICON (zHPF) is a new FICON protocol and system I/O architecture which results in improvements for small block transfers to disk using the device independent random access method.

IBM Z instructions
Db2 can take advantage of the latest IBM Z instructions. These instructions can streamline specific processes and reduce the CPU workload.

Increased System z10® page size
Db2 benefits greatly from the 1 MB page size of System z10. The increased page size allows for Db2 buffer pool enhancements which can reduce the CPU workload.

Improved hardware compression
Improved hardware compression has a positive impact on performance. For example, utilities that run against compressed data run faster.

Db2 in the z/OS environment

Db2 operates as a formal subsystem of z/OS and works efficiently with other z/OS subsystems and components.

Db2 operates as a formal subsystem of z/OS. A *subsystem* is a secondary or subordinate system that is usually capable of operating independently of, or asynchronously with, a controlling system. A Db2 subsystem is a distinct instance of a relational DBMS. Its software controls the creation, organization, and modification of a database and the access to the data that the database stores.

z/OS processes are separated into regions that are called *address spaces*. Db2 for z/OS processes execute in several different address spaces, as indicated below.

Database services address space (*ssnm*DBM1)
The *database services address space*, *ssnm*DBM1, provides most database-related services. Most large storage areas reside above the 2 GB bar in the *ssnm*DBM1 address space. With 64-bit virtual addressing to access these storage areas, Db2 can scale to extremely large sizes.

System services address space (*ssnm*MSTR)
The *system services address space*, *ssnm*MSTR, performs a variety of system-related functions.

Distributed data facility address space (*ssnm*DIST)
The *distributed data facility address space, ssnm*DIST provides support for remote requests.

IRLM (internal resource lock manager) address space
IRLMPROC controls Db2 locking.

WLM-established address spaces
Zero to many address spaces for stored procedures and user-defined functions. WLM-established address spaces are handled in order of priority and are isolated from other stored procedures or user-defined functions that run in other address spaces.

Allied agent address spaces
At least one, possibly several, of the following types of user address spaces:

- TSO
- Batch
- CICS
- IMS dependent region
- IMS control region
- WebSphere

Restriction: Db2 address spaces, and all connected address spaces, cannot be started with the with the REUSASID=YES keyword to use the z/OS reusable address space ID (ASID) capability.

Db2 works efficiently with other z/OS subsystems and components, such as the z/OS Security Server and the IBM Z Parallel Sysplex environment.

Db2 utilities run in the z/OS batch or stored procedure environment. Applications that access Db2 resources can run within the same z/OS system in the CICS, IMS, TSO, WebSphere, stored procedure, or batch environments, or on other operating systems. These applications can access Db2 resources by using the client/server services of the Db2 distributed data facility (DDF). IBM provides attachment facilities to connect Db2 to each of these environments.

Related concepts

Db2 attachment facilities
An *attachment facility* provides the interface between Db2 and another environment. You can also begin Db2 sessions from other environments on clients such as Microsoft Windows or UNIX by using interfaces like ODBC, JDBC, and SQLJ.

Distributed data facility

The *distributed data facility (DDF)* allows client applications that run in an environment that supports DRDA to access data at Db2 servers. In addition, a Db2 application can access data at other Db2 servers and at remote relational database systems that support DRDA.

Db2 in a Parallel Sysplex environment
The Parallel Sysplex is a key example of the synergy of Db2 and the IBM Z environment.

Db2 and the z/OS Security Server
The z/OS Security Server prevents unauthorized system access and can protect Db2 resources, such as tables. The z/OS Security Server is sometimes referred to as RACF, which is one of its key components.

Db2 internal resource lock manager

The Db2 internal resource lock manager (IRLM) is both a separate subsystem and an integral component of Db2. IRLM works with Db2 to control access to your data.

IRLM is shipped with Db2, and each Db2 subsystem must have its own instance of IRLM. You cannot share IRLM between Db2 subsystems or between Db2 and IMS subsystems. IRLM is also shipped with IMS. If you run a Db2 data sharing group, an IRLM group corresponds to that data sharing group.

IRLM works with Db2 to serialize access to your data. Db2 requests locks from IRLM to ensure data integrity when applications, utilities, and commands attempt to access the same data.

Recommendation: Always run with the latest level of IRLM.

IRLM requires some control and monitoring. The external interfaces to the IRLM include:

Installation
Install IRLM when you install Db2. Consider that locks take up storage, and adequate storage for IRLM is crucial to the performance of your system.

Another important performance item is to set WLM goals to maximize response time and execution velocity for the IRLM address space above all the other Db2 address spaces.

Commands
Some IRLM-specific z/OS commands enable you to modify parameters, display information about the status of the IRLM and its storage use, and start and stop IRLM.

Tracing
The Db2 trace facility enables you to trace lock interactions.

You can use z/OS trace commands or IRLMPROC options to control diagnostic traces for IRLM. You normally use these traces under the direction of IBM Support.

Related concepts
Improved performance through the use of locks
Db2 uses locks on various data objects.

Db2 and the z/OS Security Server

The z/OS Security Server prevents unauthorized system access and can protect Db2 resources, such as tables. The z/OS Security Server is sometimes referred to as RACF, which is one of its key components.

To control access to your z/OS system, you can use the Resource Access Control Facility (RACF) component of the z/OS Security Server or an equivalent product. When users begin sessions, the z/OS Security Server checks their identities to prevent unauthorized system access. The z/OS Security Server provides effective protection for Db2 data by permitting only Db2-managed access to Db2 data sets.

By using the z/OS Security Server, you can directly control most authorization to Db2 objects, define authorization, or use multilevel security.

Recommendation: Use the z/OS Security Server to check the identity of Db2 users and to protect Db2 resources.

Related concepts
Authorization and security mechanisms for data access

Authorization is an important part of controlling Db2. The security and authorization mechanisms that control access to Db2 data are both direct and indirect.

Introduction to the RACF access control module (RACF Access Control Module Guide)

Db2 and DFSMS

You can use the DFSMSdfp Storage Management Subsystem (SMS) to manage Db2 disk data sets.

The purpose of DFSMS is to automate as much as possible the management of physical storage by centralizing control, automating tasks, and providing interactive controls for system administrators. DFSMS can reduce user concerns about physical details of performance, space, and device management.

Consult with your storage administrator about using DFSMS for Db2 private data, image copies, and archive logs. Data that is especially performance-sensitive might necessitate more manual control over data set placement.

Table spaces or indexes with data sets larger than 4 GB require DFSMS-managed data sets.

Extended partitioned data sets (PDSE), a feature of DFSMSdfp, are useful for managing stored procedures that run in a stored procedures address space. PDSE enables extent information for the load libraries to be dynamically updated, reducing the need to start and stop the stored procedures address space.

Db2 attachment facilities

An *attachment facility* provides the interface between Db2 and another environment. You can also begin Db2 sessions from other environments on clients such as Microsoft Windows or UNIX by using interfaces like ODBC, JDBC, and SQLJ.

The following figure shows the z/OS attachment facilities with interfaces to Db2.

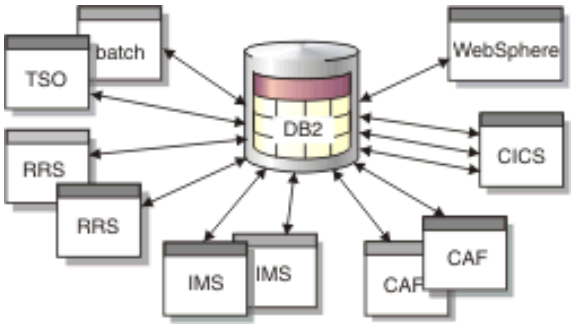


Figure 68. Attachment facilities with interfaces to Db2

The z/OS environments include:

- WebSphere
- CICS (Customer Information Control System)
- IMS (Information Management System)
- TSO (Time Sharing Option)
- Batch

The z/OS attachment facilities include:

- CICS
- IMS
- TSO
- CAF (call attachment facility)
- RRS (Resource Recovery Services)

The attachment facilities work in the various environments as follows:

- WebSphere products that are integrated with Db2 include WebSphere Application Server, WebSphere Studio, and Transaction Servers & Tools. In the WebSphere environment, you can use the RRS attachment facility.
- CICS is an application server that provides online transaction management for applications. In the CICS environment, you can use the CICS attachment facility to access Db2.
- IMS is a database computing system. IMS includes the IMS hierarchical database manager, the IMS transaction manager, and database middleware products that provide access to IMS databases and transactions. In the IMS environment, you can use the IMS attachment facility to access Db2.
- TSO provides interactive time-sharing capability from remote terminals. In the TSO and batch environments, you can use the TSO, call attachment facility (CAF), and Resource Recovery Services (RRS) attachment facilities to access Db2.
- Stored procedure environments are managed by the Workload Manager component of z/OS. In a stored procedure environment, you can use the RRS attachment facility

CICS attachment facility

The Customer Information Control System (CICS) Transaction Server provides the CICS attachment facility, which lets you access Db2 from CICS.

CICS operations, application programming, and system administration and operations organizations can use the CICS attachment facility.

CICS operations

After you start Db2, you can operate Db2 from a CICS terminal. You can start and stop CICS and Db2 independently, and you can establish or terminate the connection between them at any time. You can also allow CICS to connect to Db2 automatically.

The CICS Transaction Server also provides CICS applications with access to Db2 data while operating in the CICS environment. Any CICS application, therefore, can access both Db2 data and CICS data. In the case of system failure, CICS coordinates recovery of both Db2 data and CICS data.

The CICS attachment facility uses standard CICS command-level services where needed.

Examples::

```
EXEC CICS WAIT EXEC CICS ABEND
```

A portion of the CICS attachment facility executes under the control of the transaction issuing the SQL requests. Therefore these calls for CICS services appear to be issued by the application transaction.

With proper planning, you can include Db2 in a CICS XRF recovery scenario.

Application programming

Application programmers who write CICS command-level programs can use the same data communication coding techniques to write the data communication portions of application programs that access Db2 data. Only the database portion of the programming changes. For the database portions, programmers use SQL statements to retrieve or modify data in Db2 tables.

To a CICS terminal user, application programs that access both CICS and Db2 data appear identical to application programs that access only CICS data.

Db2 supports this cross-product programming by coordinating recovery resources with those of CICS. CICS applications can therefore access CICS-controlled resources as well as Db2 databases.

Function shipping of SQL requests is not supported. In a CICS multi-region operation (MRO) environment, each CICS address space can have its own attachment to the Db2 subsystem. A single CICS region can be connected to only one Db2 subsystem at a time.

System administration and operations

An authorized CICS terminal operator can issue Db2 commands to control and monitor both the attachment facility and Db2 itself. Authorized terminal operators can also start and stop Db2 databases.

Even though you perform Db2 functions through CICS, you need to have the TSO attachment facility and ISPF to take advantage of the online functions supplied with Db2 to install and customize your system. You also need the TSO attachment to bind application plans and packages.

IMS attachment facility

The IMS attachment facility enables you to access Db2 from IMS.

The IMS attachment facility receives and interprets requests for access to Db2 databases by using exit routines that are part of IMS subsystems. An *exit routine* is a program that runs as an extension of Db2 when it receives control from Db2 to perform specific functions. Usually, IMS connects to Db2 automatically with no operator intervention.

In addition to Data Language I (DL/I) and Fast Path calls, IMS applications can make calls to Db2 by using embedded SQL statements. In the case of system failure, IMS coordinates recovery of both Db2 data and IMS data.

With proper planning, you can include Db2 in an IMS Extended Recovery Facility (XRF) recovery scenario.

With the IMS attachment facility, Db2 provides database services for IMS dependent regions. DL/I batch support allows any authorized user to access both IMS data and Db2 data in the IMS batch environment.

Application programming, system administration, and operations organizations can use the CICS attachment facility.

Application programming

With the IMS attachment facility, Db2 provides database services for IMS dependent regions. DL/I batch support allows users to access both IMS data (DL/I) and Db2 data in the IMS batch environment, which includes:

- Access to Db2 and DL/I data from application programs.
- Coordinated recovery through a two-phase commit process.
- Use of the IMS extended restart (XRST) and symbolic checkpoint (CHKP) calls by application programs to coordinate recovery with IMS, Db2, and generalized sequential access method (GSAM) files.

IMS programmers who write the data communication portion of application programs do not need to alter their coding technique to write the data communication portion when accessing Db2; only the database portions of the application programs change. For the database portions, programmers code SQL statements to retrieve or modify data in Db2 tables.

To an IMS terminal user, IMS application programs that access Db2 appear identical to IMS.

Db2 supports this cross-product programming by coordinating database recovery services with those of IMS. Any IMS program uses the same synchronization and rollback calls in application programs that access Db2 data as they use in IMS application programs that access DL/I data.

Another aid for cross-product programming is the IMS DataPropagator licensed program, which enables automatic updates to Db2 tables when corresponding information in an IMS database is updated. This product also enables automatic updates to an IMS database when a Db2 table is updated.

System administration and operations

An authorized IMS terminal operator can issue Db2 commands to control and monitor Db2. The terminal operator can also start and stop Db2 databases.

Even though you perform Db2 functions through IMS, you need the TSO attachment facility and ISPF to take advantage of the online functions supplied with Db2 to install and customize your system. You also need the TSO attachment facility to bind application plans and packages.

TSO attachment facility

You can bind application plans and packages and run several online functions of Db2 through the TSO attachment facility. TSO also enables authorized Db2 users or jobs to create, modify, and maintain databases and application programs.

Using the TSO attachment facility, you can access Db2 by running in either foreground or batch. You gain foreground access through a TSO terminal; you gain batch access by invoking the TSO terminal monitor program (TMP) from a batch job.

Most TSO applications must use the TSO attachment facility, which invokes the DSN command processor. Two command processors are available:

DSN command processor
Provides an alternative method for running programs that access Db2 in a TSO environment. This processor runs as a TSO command processor and uses the TSO attachment facility.

Db2 Interactive (DB2I)
Consists of Interactive System Productivity Facility (ISPF) panels. ISPF has an interactive connection to Db2, which invokes the DSN command processor. Using DB2I panels, you can run SQL statements, commands, and utilities.

Whether you access Db2 in foreground or batch, attaching through the TSO attachment facility and the DSN command processor makes access easier. Together, DSN and TSO provide services such as automatic connection to Db2, attention-key support, and translation of return codes into error messages.

When using DSN services, your application must run under the control of DSN. You invoke the DSN command processor from the foreground by issuing a command at a TSO terminal. From batch, you first invoke TMP from within a batch job, and you then pass commands to TMP in the SYSTSIN data set.

After DSN is running, you can issue Db2 commands or DSN subcommands. However, you cannot issue a START Db2 command from within DSN. If Db2 is not running, DSN cannot establish a connection. A connection is required so that DSN can transfer commands to Db2 for processing.

Related concepts
DSN command processor (Db2 Application programming and SQL)
Related tasks
Issuing commands from TSO terminals (Db2 Administration Guide)
Related reference
DSN (TSO) (Db2 Commands)

Call attachment facility

The *call attachment facility (CAF)* provides an alternative connection for TSO and batch applications that need tight control over the session environment.

Applications that use CAF can explicitly control the state of their connections to Db2 by using connection functions that CAF supplies.

Resource Recovery Services attachment facility

The RRS feature of z/OS coordinates commit processing of recoverable resources in a z/OS system. Db2 supports use of these services for Db2 applications that use the RRS attachment facility (RRSAF), which Db2 provides.

The implementation of z/OS Resource Recovery Services (RRS) is based on the same technology as that of CAF but offers additional capabilities. Use the RRS attachment facility to access resources such as SQL tables, DL/I databases, MQSeries messages, and recoverable Virtual Storage Access Method (VSAM) files within a single transaction scope. Programs that run in batch and TSO can use RRSAF. You can use RRS with stored procedures and in a WebSphere environment.

The RRS attachment is required for stored procedures that run in a WLM-established address space.

Distributed data facility

The *distributed data facility (DDF)* allows client applications that run in an environment that supports DRDA to access data at Db2 servers. In addition, a Db2 application can access data at other Db2 servers and at remote relational database systems that support DRDA.

DDF supports TCP/IP and Systems Network Architecture (SNA) network protocols. DDF allows the Db2 server to act as a gateway for remote clients and servers. A Db2 server can forward requests on behalf of remote clients to other remote servers regardless of whether the requested data is on the Db2 server.

With DDF, you can have up to 150,000 connections to a single Db2 server at the same time. You can only have up to 2000 threads running concurrently. A *thread* is a Db2 structure that describes an application's connection and traces its progress.

DDF uses methods for transmitting query result tables that minimize network traffic when you access distributed data. You can also use stored procedures to reduce processor and elapsed-time costs of distributed access. A *stored procedure* is user-written SQL program that a requester can invoke at the server. When you encapsulate SQL statements to the Db2 server into a single message, many fewer messages flow across the wire.

Local Db2 applications can also use stored procedures to take advantage of the ability to encapsulate SQL statements that are shared among different applications.

In addition to optimizing message traffic, DDF enables you to transmit large amounts of data efficiently by using the full bandwidth of the network.

DDF also enables applications that run in a remote environment that supports DRDA. These applications can use DDF to access data in Db2 servers. Examples of application requesters include IBM DB2 Connect and other DRDA-compliant client products.

The decision to access distributed data has implications for many Db2 activities: application programming, data recovery, and authorization, to name a few.

Related concepts

Distributed data access

Distributed computing environments typically involve requests from users at one DBMS client that are processed by a DBMS server. The server DBMS is typically remote to the client. Certain programming techniques and performance implications apply to distributed computing environments.

Db2 in a Parallel Sysplex environment

The Parallel Sysplex is a key example of the synergy of Db2 and the IBM Z environment.

Db2 takes advantage of the Parallel Sysplex environment with its superior processing capabilities. When you have two or more processors sharing the same data, you can:

- Maximize performance while minimizing cost
- Improve system availability and concurrency
- Configure your system environment more flexibly
- Grow your system incrementally

With data sharing, applications that run on more than one Db2 subsystem can read from and write to the same set of data concurrently. This capability enables you to continuously access Db2 data, even while a node is being upgraded with new software.

Db2 subsystems that share data must belong to a Db2 *data sharing group*. A data sharing group is a collection of one or more Db2 subsystems that access shared Db2 data. Each Db2 subsystem that belongs to a particular data sharing group is a *member* of that group. All members of a group use the same shared Db2 catalog. The following figure shows an example of a data sharing group with three members.

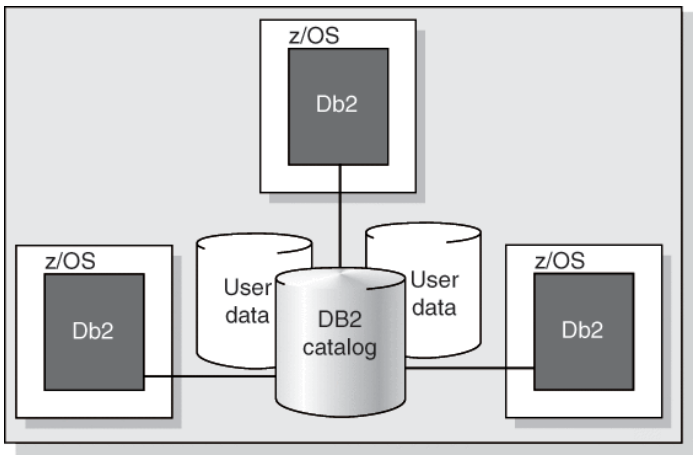


Figure 69. A Db2 data sharing group

With a data sharing group, the number of threads that can connect to a Db2 server multiplies by the number of subsystems in the group. For example, an eight-member data sharing group can have over a million simultaneous threads connect to a Db2 server.

With data sharing, you can grow your system incrementally by adding additional central processor complexes and Db2 subsystems to the data sharing group. You do not need to move part of the workload onto another system, alleviating the need to manage copies of the data or to use distributed processing to access the data.

You can configure your environment flexibly. For example, you can tailor each z/OS image to meet the requirements for the user set on that image. For processing that occurs during peak workload periods, you can bring up a dormant Db2 to help process the work.

Related concepts

Db2 data sharing

In Db2 *data sharing*, applications that reside in multiple Db2 subsystems in a Parallel Sysplex can read from and write to the same Db2 for z/OS data concurrently, with integrity, performance, scalability, and dynamic workload balancing.

Chapter 8. An overview of Db2 for z/OS and related products

A good way to start learning Db2 for z/OS is to observe how real organizations use it. Thousands of companies around the world use Db2 to run their businesses.

For you to observe even a small percentage of those businesses would be impractical. Scenarios can help you imagine some of the possibilities by describing a few ways in which organizations depend on Db2 to accomplish their business objectives.

In addition to understanding how organizations depend on Db2 to accomplish their business objectives, you also need to understand the overall IBM strategy for helping its customers effectively manage enterprise data.

You also need to understand how Db2 works with a wide variety of operating systems.

Scenarios for using Db2

Scenarios can illustrate how some organizations might successfully use Db2.

What do the following situations have in common?

- An international bank that provides uninterrupted services to its customers 24 hours a day.
- A multi-campus university system that educates thousands of students and offers hundreds of courses.
- An electric company that provides electricity to a large geographic region.

The common characteristic in each situation is that Db2 is a key ingredient in the data processing environment of each organization.

If you are new to Db2, you might wonder how these and other organizations use the product. You might wonder what types of organizations use Db2. Maybe you wonder if the organizations that use Db2 have all, or only a portion, of their data on the enterprise server. (Sometimes people refer to the enterprise server as the "mainframe.") You might wonder why organizations still continue to put their core business data on the mainframe.

Availability and scalability for large businesses

Large businesses choose Db2 for z/OS because they need a robust database server that ensures superior availability and scalability.

You might be thinking that the terms "enterprise server" and "mainframe" imply that very large businesses use a product like Db2 for z/OS.

You might ask the question: "Why do large businesses choose Db2 for z/OS?" The answer is, "Because these companies need a robust database server that ensures superior availability and scalability."

Superior availability and scalability in a Parallel Sysplex environment are the key features that distinguish Db2 for z/OS from other database servers. Because of these qualities, Db2 for z/OS is widely deployed in industries that include:

- Major credit card companies
- Banks
- Insurance companies
- Brokerage companies
- Credit information companies

These are companies that process very high volumes of transactions that require millions of concurrent updates every day.

© Copyright IBM Corp. 2001, 2020

Consider some examples.

- The volume of trading that goes on at the major stock exchanges can reach over 1,000,000,000 shares in a single day.
- A brokerage company might have a network of thousands of financial advisors and millions of customers who need online access to highly sensitive financial information daily.
- A transportation company might deliver more than 10 million packages in a single day. Each package requires several steps in the delivery process, such as pick up, transit points, and final delivery. The status of the package can be shown to customers on the web.
- A credit information company needs to provide millions of credit reports each day, while keeping the data current with more than 100 million updates in a single day.
- A sports website provides statistics, results, and live updates for their viewers. The site must continue to provide this information to their viewers quickly and efficiently during hours of peak demand.

You can easily understand why these businesses need the database system that processes these transactions to be continuously available, scalable, and secure. These enterprise systems must be available to customers who are searching for and relying on their services 24 hours a day.

- Systems must provide continuous availability.

  If you are waiting for a financial transaction to process and the application that runs that transaction suddenly fails, you might lose the opportunity to make a stock trade at a critical time. The key objective of high availability is to ensure that a system has no single point of failure.

- Systems must be scalable.

  As businesses grow, their data processing needs also grow. Business ventures, such as mergers, acquisitions, and new services, or new government regulations, can accelerate how quickly the data processing needs of the business grow. As rapid growth occurs, companies need a way to scale their business successfully.

  Companies need a large database system that is designed to easily absorb ongoing additions of new types of information and application processes without sacrificing performance or availability. That database system should never impose a constraint on growth. As businesses add more computing capacity, the database system must expand accordingly to ensure that businesses gain the full advantage of the added capacity and have continuous access to their data.

- Systems must be secure.

  The needs for protection and regulatory compliance are expanding greatly. Customers must be able to trust the database when they use it to manage valuable information such as finances or personal information. IBM Z has a long history of system integrity and security. Db2 security responds to the needs.

The following scenarios describe how a large international bank benefits from these Db2 for z/OS strengths to provide the highest quality of service to its customers.

Scenario 1: Bank mergers occur often. As two banks combine operations, how does the newly formed bank merge unrelated applications?

Db2 for z/OS data sharing in a Parallel Sysplex environment provides the solution that the new bank needs so that the two banking systems can be merged.

Parallel Sysplex clustering technology in Db2 is the answer to availability and scalability. A *Parallel Sysplex* is a *cluster*, or complex, of z/OS systems that work together to handle multiple transactions and applications. This technology implements a data sharing design.

The Db2 data sharing design gives businesses the ability to add new Db2 subsystems into a data sharing group, or cluster, as the need arises and without disruption. As applications run on more than one Db2 subsystem, they can read from and write to the same set of shared data concurrently.

The Parallel Sysplex can grow incrementally without sacrificing performance. Parallel Sysplex architecture is designed to integrate up to 32 systems in one cluster. In a shared-disk cluster, each system is a member of the cluster and has access to shared data.

An integral component of a Parallel Sysplex is the *coupling facility*, a mechanism that coordinates transactions between the different members within a cluster. Other solutions attempt to implement similar capabilities through software, but messaging by using software can cause high overhead and directly impact the ability to scale and perform.

When Parallel Sysplex technology is used, the applications from each bank can easily be integrated into a data sharing group and can access shared data.

Scenario 2: The bank runs batch jobs every night and the online workload is running close to 24 hours a day. How can the bank run varied workloads, keep them balanced, and avoid problems at peak times?

Db2 works closely with the z/OS Workload Manager (WLM) component. WLM provides the best way to run mixed workloads concurrently, and data sharing gives the bank a lot of flexibility in how to run the workloads.

Parallel Sysplex technology is designed to handle varied and unpredictable workloads efficiently. The Workload Manager ensures that the bank's workloads are optimally balanced across the systems in the Sysplex.

For example, when the bank adds a new subsystem or the workload becomes unbalanced, data does not need to be redistributed. The new subsystem has the same direct access to the data as all existing subsystems in the data sharing group.

Data sharing works with WLM to give the bank the flexibility it needs to handle peak loads easily. WLM provides the ability to start up servers and subsystems on demand, based on predefined service goals. For example, the bank can start data sharing members to handle peak loads at quarter-end processing, and stop them when the quarter-end peak finishes.

Db2 is the only data server on IBM Z to take full advantage of WLM capabilities.

Scenario 3: The bank creates a website to provide online banking to its customers 24 hours a day. Now the DBMS can never be out of service for maintenance activities. How can the bank apply maintenance to its DBMS if it needs to be operational 24 hours a day?

Data sharing and Parallel Sysplex technology give the bank a way to apply software maintenance (a planned outage) while always keeping a subset of its Db2 subsystems up and running.

The Parallel Sysplex environment provides multiple paths to data and builds redundancy into the coupling facility to avoid single points of failure. With Parallel Sysplex technology, the bank can add maintenance to one member at a time while their systems continue running and remain up-to-date on service. The technology also allows the bank to migrate to a new software release by applying the new release to one member at a time. With this design, the bank avoids outages.

In the event of an application or a system failure on one system (an unplanned outage), the Workload Manager ensures that other systems within the Sysplex can take over the full workload. Again, the bank avoids outages.

Related concepts

Db2 data sharing
In Db2 *data sharing*, applications that reside in multiple Db2 subsystems in a Parallel Sysplex can read from and write to the same Db2 for z/OS data concurrently, with integrity, performance, scalability, and dynamic workload balancing.

Db2 in a Parallel Sysplex environment
The Parallel Sysplex is a key example of the synergy of Db2 and the IBM Z environment.

Critical business information for decision makers

Most organizations use various hardware and software products to store a large amount of data. Db2 can assist in providing essential information to key decision makers that helps them to make critical business decisions.

Consider a multi-campus university system. A group of educational experts manages the system from day to day. These people make decisions that affect all the university campuses. The decision makers use a

data warehouse so that they can "mine" data from system databases and make the best organizational decisions.

You can think of a *data warehouse* as a system that provides critical business information to an organization. *Data mining* is the act of collecting critical business information from that data warehouse, correlating it, and uncovering associations, patterns, and trends. The data warehouse system cleanses the data for accuracy and currency. The data warehouse system also presents the data to the decision makers so that they can interpret and use it effectively and efficiently.

Data warehousing and data mining are related terms that are encompassed by the more global term, *business intelligence*.

Most organizations use various hardware and software products to store a large amount of data. However, many key decision makers do not have timely access to the information that they need to make critical business decisions. If they had the information, they could make more intelligent decisions for their businesses—thus, the term business intelligence.

The university's data warehouse system, which relies on Db2, transforms the vast amount of data from being operational to being informational. An example of operational data in a university is the identities of people who enroll in various classes. Clearly, the university needs this information to operate. This operational data becomes informational when, for example, decision makers discover that most students who enroll in Advanced Calculus also enroll in Music Appreciation. The university does not require this information to operate, but decision makers can run a more effective institution if they have informational data. As a result of having access to this informational data, university personnel can make better decisions. Individuals who plan class schedules can ensure that these classes do not meet at the same time, enabling students to enroll in both classes. Using Db2 as your enterprise data warehouse ensures that you are making key business decisions based on data that is correct.

The university also uses the power of the Internet. Each campus has a website, which supplies relevant information to university decision makers, students, parents, and members of the communities that surround each campus.

Using Db2 for z/OS as its enterprise server, the university can act as follows:

- Evaluate the effectiveness of curriculum, expenditures, professors, and professional development
- Identify emerging trends early enough for effective action
- Complete grant applications more quickly and effectively
- Compile a complete summary report on any individual student
- Enable authorized users to use the web to perform any of these actions, plus others

Data distribution and Web access

The ability to distribute data and provide Web access to that data is vital to service providers and their customers.

An electric company provides electricity to a large geographic region. Working out of a single office, the company's customer service representatives answer customer calls and submit requests for service. The electric company has hundreds of field representatives who provide service at customer locations. The field representatives work out of many local offices, and they need access to customer service requests that the central office receives.

The customer service representatives document customer requests on their workstations, which have DB2 Connect Personal Edition. This information is uploaded to Db2 for z/OS. The field representatives can then use Java applications to access the customer request information in Db2 from their local offices.

In this scenario, the electric company's distributed environment relies on the distributed data facility (DDF), which is part of Db2 for z/OS. Db2 applications can use DDF to access data at other Db2 sites and at remote relational database systems that support *Distributed Relational Database Architecture* (DRDA). *DRDA* is a standard for distributed connectivity. An organization called The Open Group developed the standard, with active participation from many companies in the industry, one of which was IBM. All IBM Db2 data servers support this DRDA standard.

DDF also enables applications that run in a remote environment that supports DRDA. These applications can use DDF to access data in Db2 servers. Examples of application requesters include IBM DB2 Connect and other DRDA-compliant client products.

Related concepts

Db2 and the web
Db2 provides many benefits to companies that operate on the web.

Distributed data facility
The *distributed data facility (DDF)* allows client applications that run in an environment that supports DRDA to access data at Db2 servers. In addition, a Db2 application can access data at other Db2 servers and at remote relational database systems that support DRDA.

Db2 data servers and environments

Db2 data server products run on a wide set of operating systems, including z/OS, IBM i, Linux, UNIX, and Windows.

In addition to learning about Db2 for z/OS, you will also want to know about some of the other products that work with Db2 for z/OS. Your company probably uses some of these other products.

Db2 data servers include support for the following products:

- Db2 for z/OS
- DB2 for i
- Db2 for Linux, UNIX, and Windows
- Db2 for Linux on IBM Z

Recommendation: Download free or trial demonstration versions of many Db2 products and tools. By using demonstration code, you can increase your understanding of the various products that you will read about in this information. To download demonstration copies, visit the IBM software downloads web page. From that page, you can select a specific Db2 product, and choose the download option on that product's home page.

IBM specifically developed the Db2 data servers so that the underlying code of each DBMS uses the individual capabilities of the various operating systems.

The Db2 data server products encompass the following characteristics:

- Data types among the Db2 data servers are compatible.
- Open standards mean that many different types of clients can access data in the Db2 data servers.
- You can develop applications with SQL that are common across Db2 data servers and *port* them from one Db2 operating system to another with minimal modification. (Porting means moving an application from one operating system to another.)
- Db2 data servers can support applications of any size. For example, imagine that your application starts with a small number of users and small volumes of data and transactions, but then it grows significantly. Because of compatibility across Db2 data servers, your application can continue to work efficiently as you transition to IBM Z.
- Similar function is typically incorporated into each Db2 data server over time.
- Tools are available to help you manage all the Db2 data servers in a similar way.

Tip: Identify someone who is familiar with your company's I/S environment. Ask that person to provide a list of the products that you will likely work with. Your company might have only a subset of the products that are mentioned in this information. Knowing basic information about your company's environment will help you know which topics are most important for you to read.

Enterprise servers

Enterprise servers are the systems that manage the core business data across an enterprise and support key business applications.

The following Db2 products can act as enterprise servers:

- Db2 for z/OS
- Db2 for Linux, UNIX, and Windows
- DB2 for i, which supports applications in the midrange IBM i environment
- DB2 for VSE & VM, supporting large applications on the VSE and VM environments

Related concepts

z/Architecture and the z/OS operating system
z/OS, which is highly secure, scalable, and open, offers high-performance that supports a diverse application execution environment. The tight integration that Db2 has with the IBM Z architecture and the z/OS environment creates a synergy that allows Db2 to exploit advanced z/OS functions.

Db2 distributed editions

Several Db2 family products run in distributed or workstation environments.

For more information about Db2 for Linux, UNIX, and Windows and distributed Db2 database product editions, see Db2 database product editions and Db2 offerings .

Related reference

Db2 for Linux, UNIX, and Windows

Db2 on smaller-scale servers

In addition to the enterprise servers, most companies support smaller-scale servers on local area networks (LANs). Smaller-scale servers handle important applications that don't demand the resources that are available on the larger enterprise servers.

Db2 runs on the Linux operating system, including Linux on IBM Z. The IBM Z platform offers four operating systems on which you can run Db2 data server products: z/OS, Linux, VM, and VSE.

Many customers use Db2 on Linux, on IBM Z, as their application server, connecting with Db2 for z/OS as the data server, so that they can take advantage of distributed connections and HiperSockets for fast and secure communication.

Personal, mobile, and pervasive environments

Db2 is available on small devices that are designed for individual use. You can write programs that access Db2 data on your own desktop, laptop, or handheld computer while you are traveling or working at home. Then, later you can synchronize these databases with corporate databases in the enterprise.

In the desktop and laptop workstation environments, Db2 Express provides a data server engine for a single user. Db2 Express serves your needs if you are working independently and occasionally connected or mobile. You can download and deploy Db2 Express-C for free.

For handheld computers, Db2 Everyplace® enables lightweight database applications on all the Palm Operating System, Windows CE, Embedded Linux, QNX Neutrino, Linux, and Symbian EPOC operating systems. Db2 Everyplace is available in two editions: Enterprise Edition and Database Edition. A trial version of Db2 Everyplace is available for download.

Multiple transaction and application environments

To optimize performance, throughput, and response time, organizations can distribute their application transactions and data, and they can run database queries in parallel.

A *cluster* is a complex of machines that work together to handle multiple transactions and applications. The following Db2 data server products use cluster technology:

- Db2 for z/OS

- DB2 for i, which runs in the parallel System i® environment
- Db2 for Linux, UNIX, and Windows

Db2 data server products can operate in clusters in the following environments:

- AIX
- HP-UX
- IBM i
- Linux
- Solaris
- Windows
- z/OS

Db2 and network communication

The Db2 data server products can communicate by using both wide area networks (WANs) and local area networks (LANs).

WAN
A wide area network generally supports the enterprise servers such as Db2 for z/OS; they require either Transmission Control Protocol/Internet Protocol (TCP/IP) or Systems Network Architecture (SNA).

LAN
A local area network generally supports smaller servers, which requires TCP/IP.

Clients supported by Db2 data servers

Db2 data servers support a wide variety of clients, languages, and tools.

Environments

- AIX
- Eclipse
- HP-UX
- Linux
- Solaris
- Windows
- Web browsers

Programming languages

You can write application programs for Db2 for z/OS data servers in any of the following programming languages:

- APL2
- Assembler
- C and C++
- C# “2” on page 304 (through .NET)
- COBOL
- Fortran
- High-level Assembler (or HLASM)
- Java “1” on page 304
- Perl “2” on page 304
- PHP “2” on page 304

- PL/I
- Python [“2” on page 304]
- REXX
- Ruby on Rails [“2” on page 304]
- SQL procedural language
- Visual Basic [“2” on page 304]

1. By connecting to Db2 for z/OS using IBM Data Server Driver for JDBC and SQLJ.
2. By remote connections to Db2 for z/OS using Db2 for Linux, UNIX, and Windows client drivers.

Development tools

- IBM Optim™ Development Studio
- Rational Developer for IBM Z
- Java Virtual Machine

Related concepts

Tools and IDEs for developing Db2 applications

Application developers can use various application development tools and integrated development environments (IDEs) to create database applications, and stored procedures that access and use data in Db2 for z/OS.

Sources of data

Access to heterogeneous data is a powerful asset for any organization that has data in various sources.

Db2 for Linux, UNIX, and Windows supports access to many different data sources with a single SQL statement. This support is called *federated database support*, which is provided by InfoSphere® Information Integration products. For example, with federated database support, you can join data from a wide variety of data sources. The application (and the application developer) does not need to understand where the data is or the SQL differences across different data stores. Federated data support includes support for the following relational and nonrelational data sources:

- All Db2 data server products
- IMS
- Informix
- Oracle
- Microsoft SQL Server, Microsoft Excel
- Sybase
- JDBC
- Databases that supports JDBC API
- OLE DB
- Teradata
- EMC Documentum

If you also use InfoSphere Federation Server, your applications that access the Db2 DBMS can have read-write access to additional data sources, web services, and WebSphere Business Integration. Access to heterogeneous, or dissimilar, data means that applications can accomplish more, with less code. The alternative would be that programmers would write multiple programs, each of which accesses data in one of the sources. Then the programmers would write another program that would merge the results together.

Tools for managing Db2 for z/OS

Many different products and tools are available in the marketplace to help you manage your Db2 environments, regardless of which operating system you use.

Db2 Tools for z/OS are helpful to people who are managing a Db2 environment:

Db2 Tools for z/OS

Db2 Tools for z/OS products support the following capabilities:

Database administration
Navigate through database objects and perform database administration tasks on one or many objects at a time. This category also includes tools that are used to alter, migrate, and compare objects in the same or in different Db2 systems.

Utility management
Manage Db2 systems with high-performance utilities and automation.

Performance management
Monitor and tune Db2 systems and applications to obtain optimal performance and lowest cost.

Recovery management
Examine recovery assets and recover Db2 objects to a point in time in the event of system outage or application failure. This category also includes tools to help you manage recovery assets.

Replication management
Propagate data changes by capturing and applying changes to remote systems across the Db2 data servers.

Application management
Manage Db2 application changes with minimal effort, and build and deploy applications across the enterprise.

Many of the tools that support Db2 for z/OS provide both a graphical user interface (GUI) and an ISPF interface that allows you to perform most Db2 tasks interactively. With the ISPF interfaces integrated together, you can move seamlessly from one tool to another.

Related concepts

Tools and IDEs for developing Db2 applications
Application developers can use various application development tools and integrated development environments (IDEs) to create database applications, and stored procedures that access and use data in Db2 for z/OS.

Related reference

Db2 Tools for z/OS

Facilities and tools for Db2 performance monitoring (Db2 Performance)

Related information

Db2 Tools for z/OS Product Documentation

IBM Data Server Manager

IBM Data Server Manager is an integrated database management tools platform that is used for Db2 for z/OS subsystems. With Data Server Manager, you can monitor performance, track configuration changes, administer databases, and optimize the database environment.

Database administration

You can use Data Server Manager for the following database administration tasks:

- Managing database objects and changes
- Developing and running SQL scripts
- Job scheduling to automate changes to a database

Query analysis and performance optimization

You can use Data Server Manager for the following query optimization tasks:

- Tuning individual or entire workloads
- Receiving expert recommendations from tuning advisors about statistics, indexes, and performance
- Capturing SQL and workloads from various sources (for example, performance monitor, development environment, or directly from the database)
- Viewing advanced access plan visualization (graph or grid based)
- Using wizards to identify tuning opportunities and assess benefits and impact
- Comparing SQL performance over time by accessing real-time and historical performance metrics
- Creating reports to share findings or as input for trend analysis and capacity planning
- Resolving availability and performance problems in your system based on alerts

Configuration management

You can use Data Server Manager for the following configuration management tasks:

- Tracking configuration changes for monitored databases
- Comparing configurations between two databases
- Cloning configurations from one database to another
- Getting insights into IT deployments
- Enforcing database connection properties from a central location
- Redirecting database access to another database

Autonomic management

You can use Data Server Manager for the following autonomic management tasks:

- View and manage the overall health of Db2 for z/OS systems across the enterprise
- Access identified Db2 for z/OS symptoms and recommended actions
- View resource dashboards that consolidate information from various sources and tools with data sharing groups and individual table space and index space partitions
- Pinpoint and navigate to resources with the highest severity symptoms and actions

Storage optimization

You can use Data Server Manager for the following storage optimization tasks:

- Review storage accesses over time
- Identify how often instances, schemas, tables, indexes, and table spaces are accessed or modified
- View details for historical accesses and take suggested actions (for example, configuring adaptive or static compression to realize potential storage savings)

Related reference

Db2 Tools for z/OS

Related information

IBM Data Server Manager (product home page)

Overview of Data Server Manager

IBM Data Server Manager Version 2.1.x releases: new features and enhancements

IBM Data Studio

IBM Data Studio is a set of powerful set of tools that help you manage enterprise data, databases, and data-driven applications.

Tip: IBM Data Server Manager is the recommended replacement tool for many of the capabilities in IBM Data Studio. For more information, see "IBM Data Server Manager" on page 305.

IBM Data Studio includes the following tools:

- IBM InfoSphere Data Architect
- Optim Database Relationship Analyzer
- IBM Data Studio
- IBM Optim pureQuery Runtime
- IBM Data Studio Administrator
- IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS
- IBM Database Encryption Expert
- Db2 High Performance Unload for z/OS

IBM Data Studio can help you perform a wide range of daily activities. You can use IBM Data Studio to manage Db2 databases on different operating systems.

Database administration

You can use IBM Data Studio to administer Db2 instances, Db2 for z/OS subsystems, databases, and database objects from workstation computers. You can also run utilities that reorganize or load your data in your existing Db2 for z/OS databases.

Application development and deployment

You can use IBM Data Studio for the following application development tasks:

- Developing pureQuery applications in a Java project
- Creating, testing, debugging, and deploying routines, such as stored procedures and user-defined functions
- Creating, editing, and running SQL queries
- Connecting to data sources and browsing data objects and their properties
- Creating and altering data objects

Query analysis and performance optimization

IBM Data Studio provides query serviceability and tuning capabilities from workstation computers for queries that run in Db2 for z/OS environments. You can use IBM Data Studio to for the following query optimization tasks:

- View query activity and identify problematic queries.
- Get tuning recommendations for statistics that could improve query performance.
- Graph the access path for a query.
- Graphically generate optimization hints.
- Generate reports with information about tables and predicates that are associated with a particular query.
- View the values of subsystem parameters.

Related reference

IBM Data Studio product documentation (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect)

Related information

IBM Data Studio (product home page)

IBM Data Studio and InfoSphere Optim pureQuery Runtime (IBM developerWorks)

Db2 Administration Tool for z/OS

The Db2 Administration Tool for z/OS simplifies many of the administrative tasks that are required to maintain your Db2 subsystem.

You can use this tool to perform the following tasks:

- Manage your Db2 environments efficiently with a comprehensive set of functions
- Display and interpret objects in the Db2 catalog and perform catalog administration tasks
- Change and update presented data quickly and easily
- Use alter and migrate functions

Related reference

Db2 Administration Tool for z/OS

The Db2 command line processor

You can use the Db2 command line processor to issue SQL statements, bind DBRMs that are stored in HFS files, and call stored procedures.

The Db2 command line processor on Db2 for z/OS is a Java application that runs under z/OS UNIX System Services. The Db2 command line processor automatically directs output to the standard output device, and notifies the user of successful or unsuccessful commands.

Related concepts

The Db2 command line processor (Db2 Commands)

Related tasks

Running stored procedures from the command line processor (Db2 Application programming and SQL)

Tools and IDEs for developing Db2 applications

Application developers can use various application development tools and integrated development environments (IDEs) to create database applications, and stored procedures that access and use data in Db2 for z/OS.

The following application development tools and IDEs support the development of Db2 for z/OS database applications and routines.

IBM Data Studio

IBM Data Studio is a suite of Eclipse-based tools that are for development database administrators and application developers. You can use IBM Data Studio for the following tasks:

- Developing pureQuery applications in a Java project
- Creating, testing, debugging, and deploying routines, such as stored procedures and user-defined functions
- Creating, editing, and running SQL queries
- Connecting to data sources and browsing data objects and their properties
- Creating and altering data objects

Rational Developer for System z

Rational Developer for System z can improve efficiency and helps with mainframe development, web development, and integrated mixed workload or composite development. By using Rational Developer for System z, you can accelerate the development of your web applications, traditional COBOL and PL/I applications, web services, and XML-based interfaces.

Rational Developer for System z provides a common workbench and an integrated set of tools that support end-to-end, model-based application development, run time testing, and rapid deployment of On Demand applications. With the interactive, workstation-based environment, you can quickly access your z/OS data.

Rational Application Developer for WebSphere Software

IBM Rational software provides a full range of tools to meet your analysis, design, and construction needs, whether you are an application developer, application architect, systems engineer, or database designer. IBM Rational Application Developer for WebSphere Software helps developers to quickly design, develop, analyze, test, profile, and deploy high-quality web, Service-oriented Architecture (SOA), Java, J2EE, and portal applications.

By using Rational Application Developer, you can increase productivity, minimize your learning curve, and shorten development and test cycles so that you can deploy applications quickly.

WebSphere Studio Application Developer

WebSphere Studio Application Developer is a fully integrated Java development environment. Using WebSphere Studio Application Developer, you can build, compile, and test J2EE (Java 2 Enterprise Edition) applications for enterprise On Demand business applications with:

- JSP (JavaServer Pages) files
- EJB (Enterprise JavaBeans) components
- 100% Pure Java applets and servlets

For more information see, IBM WebSphere Application Server Developer Tools for Eclipse.

Db2 Development add-in for Visual Studio .NET

You can use the Db2 Development add-in for Microsoft Visual Studio .NET to enhance integration with the Microsoft Visual Studio .NET development environment.

The add-in features make it easy for application programmers to work with Db2 servers and to develop Db2 routines and objects.

The key add-in features enable developers to perform the following tasks:

Building Db2 server-side objects

Db2 Connect provides a Db2 .NET Data Provider, which enables .NET applications to access Db2 for z/OS and workstation (Windows, UNIX, and Linux) operating systems.

Using the Solution Explorer, developers can use script files for building objects that include routines, triggers, tables, and views.

Accessing and manage Db2 data connections

The IBM Explorer provides access to IBM database connections and enables developers to perform the following tasks:

- Work with multiple Db2 connections
- View object properties
- Retrieve and update data from tables and views
- View source code for Db2 procedures and functions
- Generate ADO .NET code using a drag-and-drop technique

For information about using ADO .NET for applications that connect to Db2 for z/OS, see the Db2 for Linux, UNIX, and Windows information in IBM Knowledge Center .

Launching Db2 development and administration tools

These tools include IBM Data Studio, IBM Data Server Manager, Replication Center, Command Center, Task Center, and Journal.

ODBC based development tools

A wide variety of tools are available for performing tasks such as querying a database. These tools include ODBC-based tools such as Lotus Approach, Microsoft Access, Microsoft Visual Basic, Microsoft Excel, and many others. The ODBC-based tools provide a simpler alternative to developing applications than using a high-level programming language. QMF for Workstation provides access to Db2 data for these tools. With all of these tools, you can specify Db2 for z/OS as the database to access.

Related concepts

Use of development tools to create a stored procedure
Workstation-based development tools can help you create, install, and test stored procedures for Db2 for z/OS.

Middleware components

Middleware and client application programming interfaces (APIs) complement the Db2 data server products. Middleware and client APIs help Db2 products to communicate and work together more effectively.

IBM middleware components include a broad portfolio of WebSphere products that help you achieve the promise of on demand business. The product families that comprise the WebSphere portfolio provide all the infrastructure software that you need to build, deploy, and integrate your on demand business. The WebSphere products fall into the following categories:

- **Foundation & Tools** for developing and deploying high-performance business applications
- **Business Portals** for developing scalable enterprise portals and enabling a single point of personalized interaction with diverse business resources
- **Business Integration** for end-to-end application integration

WebSphere products run on the most popular operating systems, including z/OS, AIX, Linux, OS/390, IBM i, Windows, and Solaris.

IBM Rational Portfolio Manager

IBM Rational Portfolio Manager can help you align your IT and systems investments with your business goals.

IBM Rational Portfolio Manager is integrated with the following software:

- IBM Rational ProjectConsole software
- IBM Rational Method Composer software
- IBM Rational ClearQuest® software
- IBM Rational RequisitePro® software

Db2 Connect

Db2 Connect leverages your enterprise information regardless of where that information is. Db2 Connect gives applications fast and easy access to existing databases on IBM enterprise servers. The applications can be on demand business applications or other applications that run on UNIX or Microsoft Windows operating systems.

Db2 Connect offers several editions that provide connectivity to host and IBM i database servers. Db2 Connect Personal Edition provides direct connectivity, whereas Db2 Connect Enterprise Edition provides indirect connectivity through the DB2 Connect server.

With Db2 Connect, you can accomplish the following tasks:

- Extend the reach of enterprise data by providing users with fast and secure access to data through intranets or through the public Internet
- Integrate your existing core business applications with new, Web-based applications that you develop

- Create on demand business solutions by using the extensive application programming tools that come with DB2 Connect
- Build distributed transaction applications
- Develop applications by using popular application programming tools such as Visual Studio .NET, ActiveX Data Objects (ADO), OLE DB, and popular languages such as Java, PHP, and Ruby on Rails
- Manage and protect your data
- Preserve your current investment in skills

Users of mobile PCs and pervasive computing devices can use Db2 Connect to access reliable, up-to-date data from z/OS and IBM i database servers.

Db2 Connect provides the required performance, scalability, reliability, and availability for the most demanding applications that your business uses. Db2 Connect runs on AIX, HP-UX, Linux, Solaris, and Windows.

Related reference

Db2 Connect

WebSphere Application Server

WebSphere Application Server is part of the Foundation & Tools WebSphere portfolio. This product enables organizations to move quickly from simple web publishing to secure on demand business.

WebSphere Application Server is a Java 2 Enterprise Edition (J2EE) and web services technology-based platform. With WebSphere Application Server, you can take advantage of the following services:

Web services
Web services can help you develop applications more quickly.

Dynamic application services
Dynamic application services let you manage your on demand business environment with web services and J2EE 1.3 support that uses standard, modular components to simplify enterprise applications.

Integrated tools support
WebSphere Studio Application Developer provides support with integrated tools.

Related concepts

SOA, XML, and web services
XML data is a key ingredient for solutions that are based on service-oriented architecture (SOA). You can leverage XML-based SOA applications to build XML-based web services.

WebSphere Studio

WebSphere Studio is part of the Foundation & Tools WebSphere portfolio. WebSphere Studio is actually a suite of tools that spans development for the web, the enterprise, and wireless devices.

The WebSphere Studio suite of tools provides the following support:

- **For application development**: WebSphere Studio Application Developer works with Java and J2EE applications and other tools that include WebSphere Studio Enterprise Developer for developing advanced J2EE and web applications.
- **For application connectivity**: WebSphere MQ is a message handling system that enables applications to communicate in a distributed environment across different operating systems and networks.
- **For web development**: WebSphere Studio Homepage Builder is an authoring tool for new web developers, and WebSphere Studio Site Developer is for experienced web developers.

Related concepts

Web-based applications and WebSphere Studio Application Developer

The WebSphere Studio Application Developer offers features that developers can use to create web-based applications.

WebSphere Host Integration

WebSphere Host Integration is part of the Foundation & Tools WebSphere portfolio. WebSphere Host Integration provides support for applications that rely on both the web and host environments.

WebSphere Host Integration is actually a portfolio of products that help organizations access, integrate, and publish host information to web-based clients and applications.

Federated database support through WebSphere Information Integrator

The WebSphere Information Integration family of products is a key part of the information integration framework. The product components include a federated data server and a replication server for integrating these diverse types of data.

Information integration technology provides access to diverse, distributed data. This technology lets you integrate a wide range of data, including traditional application sources as well as XML, text documents, web content, email, and scanned images.

The following key technologies provide Information integration:

- Support for accessing XML data sources
- Web services support
- Federation technology
- Additional features such as advanced search and flexible data replication

The IBM federated database systems offer powerful facilities for combining information from multiple data sources. These facilities give you read and write access to diverse data from a wide variety of sources and operating systems as though the data is a single resource. With a federated system, you can:

- Keep data where it resides rather than moving it into a single data store
- Use a single API to search, integrate, and transform data as though it is in a single virtual database
- Send distributed requests to multiple data sources within a single SQL statement

For example, you can join data that is located in a Db2 table, an Oracle table, and an XML tagged file.

The IBM product that supports data federation is WebSphere Information Integrator.

Consider federation as an integration strategy when the technical requirements of your project involve search, insert, update, or delete operations across multiple heterogeneous, related sources or targets of different formats. During setup of the federated systems, information about the data sources (for example, the number and the data type of columns, the existence of an index, or the number of rows) is analyzed by Db2 to formulate fast answers to queries. The query optimization capability of federated systems can automatically generate an optimal plan based on many complex factors that are in this environment. This automatically generated plan makes application development in a federated system much easier, because developers no longer need to dictate the execution strategies in the program.

Data replication through InfoSphere Replication Server

InfoSphere Replication Server for z/OS provides high-volume, low-latency replication for business continuity, workload distribution, or business integration scenarios.

Data replication is the process of maintaining a defined set of data in more than one location. Replication involves copying designated changes from one location (a source) to another location (a target) and synchronizing the data in both locations. The source and the target can be in servers that are on the same machine or on different machines in the same network.

You can use InfoSphere Replication Server to help maintain your data warehouse and facilitate real-time business intelligence. InfoSphere Replication Server provides the flexibility to distribute, consolidate, and synchronize data from many locations by using differential replication or ETL.

InfoSphere Replication Server supports the following features:

- Queue-based and SQL-based replication models
- Data sharing configurations for Db2 for z/OS
- High-volume and low-latency data replication

WebSphere DataStage

IBM WebSphere DataStage® provides the capability to perform extract, transform, and load (ETL) operations from multiple sources to multiple targets, including Db2 for z/OS.

This ETL solution supports the collection, integration, and transformation of large volumes of data, with data structures ranging from simple to highly complex. WebSphere DataStage manages data that arrives in real time and data received on a periodic or scheduled basis.

ETL operations with WebSphere DataStage are log-based and support a broad data integration framework. You can perform more complex transformations and data cleansing, and you can merge data from other enterprise application software brands, including SAP, Siebel, and Oracle.

WebSphere QualityStage

IBM WebSphere QualityStage® provides a data quality solution that you can use to standardize customer, location, and product facts.

You can use WebSphere QualityStage to validate global address information and international names and other customer data, including phone numbers, email addresses, birth dates, and descriptive comments, to discover relationships. WebSphere QualityStage delivers the high-quality data that is required for success in a range of enterprise initiatives, including business intelligence, legacy consolidation, and master data management.

Client application programming interfaces

Application programming interfaces provide various ways for clients to access a Db2 database server.

Java interfaces

Db2 provides two standards-based Java programming application programming interfaces (APIs) for writing portable application programs that access Db2:

pureQuery
Developers can use pureQuery to build applications with less code than JDBC, but with greater control over database access than object-relational frameworks. Developers can use SQL for in-memory collections and databases without learning a new query language that is not optimal for data access.

JDBC
A generic interface for writing platform-independent applications that can access any SQL database.

SQLJ
Another SQL model that a consortium of major database vendors developed to complement JDBC. ISO (International Standards Organization) defines SQLJ. SQLJ is easier to code than JDBC and provides the superior performance, security, and maintainability of static SQL.

With Db2 for z/OS support for JDBC, you can write dynamic SQL applications in Java. With SQLJ support, you can write static SQL applications in Java. These Java applications can access local Db2 data or remote relational data on any server that supports DRDA.

With Db2 for z/OS, you can use a stored procedure that is written in Java. (The Db2 Database family supports stored procedures that are written in many additional languages.) A *stored procedure* is a user-written application program that the server stores and executes. A single SQL CALL statement invokes a stored procedure. The stored procedure contains SQL statements, which execute locally at the server. The result can be a significant decrease in network transmissions.

You can develop Java stored procedures that contain either static SQL (by using SQLJ) or dynamic SQL (by using JDBC). You can define the Java stored procedures yourself, or you can use IBM Data Studio and WebSphere Studio Application Developer tools.

ODBC

Db2 Open Database Connectivity (ODBC) is the IBM callable SQL interface for relational database access. Functions are provided to application programs to process dynamic SQL statements. Db2 ODBC allows users to access SQL functions directly through a call interface. Through the interface, applications use procedure calls at execution time to connect to databases, to issue SQL statements, and to get returned data and status information. The programming languages that support ODBC are C and C++.

Web services

Web services are self-contained, modular applications that provide an interface between the provider and consumer of On-Demand business application resources over the Internet. Web services client applications can access a Db2 database.

Db2 Database Add-ins for Visual Studio

The IBM Db2 Database Add-ins for Microsoft Visual Studio is a set of tightly integrated application development and administration tools designed for Db2 Database. The Add-ins integrate into the Visual Studio .NET development environment so that application programmers can easily work within their Integrated Development Environment (IDE) to access Db2 data.

The following features offer key benefits:

- Support for client applications (both desktop and web-based applications) to use .NET to access remote Db2 servers
- A tool for building stored procedures that makes it easy for any application programmer to develop and test stored procedures with Db2 for z/OS without prior IBM Z skills or knowledge

Related concepts

Tools and IDEs for developing Db2 applications
Application developers can use various application development tools and integrated development environments (IDEs) to create database applications, and stored procedures that access and use data in Db2 for z/OS.

Use of an application program as a stored procedure
A stored procedure is a compiled program that can execute SQL statements.

Distributed data access
Distributed computing environments typically involve requests from users at one DBMS client that are processed by a DBMS server. The server DBMS is typically remote to the client. Certain programming techniques and performance implications apply to distributed computing environments.

Use of Java to execute static and dynamic SQL
Db2 for z/OS supports SQLJ and JDBC. In general, Java applications use SQLJ for static SQL, JDBC for dynamic SQL, and PureQuery for both static and dynamic SQL.

Programming languages and methods for developing application programs that access Db2 for z/OS
You can use a wide variety of programming languages and techniques to develop application programs for Db2 for z/OS. In addition, several methods are available for communicating with Db2.

Use of ODBC to execute dynamic SQL
Open Database Connectivity (ODBC) lets you access data through ODBC function calls in your application. The ODBC interface eliminates the need for precompiling and binding your application and increases the portability of your application.

SOA, XML, and web services

XML data is a key ingredient for solutions that are based on service-oriented architecture (SOA). You can leverage XML-based SOA applications to build XML-based web services.

Open standards

Open standards provide a framework for on demand business that is widely accepted across the computer industry. With common standards, customers and vendors can write application programs that can run on different database systems with little or no modification. Application portability simplifies application development and ultimately reduces development costs.

IBM is a leader in developing open industry standards for database systems. For example, Db2 for z/OS is developed based on the following standards:

- The SQL ANSI/ISO standard
- The Open Group Technical Standard DRDA Version 3
- The JDBC API 3.0 Specification, developed by the Java Community Process

For a more complete list, see Industry standards and Db2 for z/OS (Db2 Home)

Related concepts

SQL: The language of Db2 (Db2 SQL)

Db2 and the web

Db2 provides many benefits to companies that operate on the web.

The web changed the way that companies conduct business. Corporations, both large and small, use websites to describe the services and products they provide. Shipping companies enable customers to track the progress of their shipments online. Bank customers can view their accounts and initiate online transactions from the comfort of their homes. Companies routinely distribute information about company programs, policies, and news, by using company-wide intranets. Individual investors submit online buy and sell orders through their brokerages every day. Online retailing continues to increase in popularity. Buyers use specialized software for the following types of business-to-business transactions:

- Track procurement activity
- Intelligently select preferred suppliers
- Electronically initiate business-to-business transactions with suppliers

These are just a few examples of the many ways that businesses are benefitting from the power of the web by transforming themselves into On-Demand businesses.

The world of On-Demand business might seem a bit like a jigsaw puzzle. Before you work on a puzzle, you want to know what the picture on the puzzle should look like when you are finished. Likewise, before building or working on an On-Demand business application, you must have a high-level understanding of the overall environment. You must also know something about the various products and tools in that environment. Developing and implementing your application probably involves products and tools on more than one operating system (such as z/OS, Linux, and Windows operating systems).

You can use the following products, tools, and languages in an e-business environment:

- Rational product family
- IBM Data Studio
- IMS
- Db2 product family
- CICS
- Web services
- Web browsers
- WebSphere product family, including WebSphere Information integration products
- Db2 Database Add-ins for Visual Studio

- Languages: C, C++, C#, COBOL, Java, .NET, PHP, Perl, PL/I, Python, Ruby on Rails, TOAD, and Visual Basic

Access to data is central to the vast majority of On-Demand business applications. Likewise, the business logic, which transforms data into information or which defines a business transaction, is another key component. Many organizations already store a large amount of mission-critical data in Db2 for z/OS. They also typically have a considerable investment in application programs that access and manipulate this data. Companies that are thinking about moving parts of their business to the web face the challenge of determining how to build on their existing base of data and business logic and how to expand the usefulness of this base by using the web.

The IBM premier application server, WebSphere Application Server, helps companies to enable their data and business logic for the web. WebSphere Application Server supports server-side programming, which you will learn more about in this information.

By using web-based products and tools, companies can build, deploy, and manage portable On-Demand business applications.

SOA, XML, and web services

XML data is a key ingredient for solutions that are based on service-oriented architecture (SOA). You can leverage XML-based SOA applications to build XML-based web services.

Web services are sets of business functions that applications or other web services can invoke over the Internet. A web service performs a useful service on behalf of a requester. That service can span across many businesses and industries.

Example: Assume that an airline reservation system is a web service. By offering this service, the airline makes it easier for its customers to integrate the service into their travel-planning applications. A supplier can also use the service to make its inventory and pricing information accessible to its buyers.

Web services let you access data from a variety of databases and Internet locations. Db2 can act as a web services requester, enabling Db2 applications to invoke web services through SQL. Db2 can also act as a web services provider through Db2 WORF (web services object run time framework), in conjunction with WebSphere Application Server, enabling you to access Db2 data and stored procedures as web services.

The functions that web services perform can be anything from simple requests to complicated business processes. You can define a basic web service by using standard SQL statements.

Using XML for data exchange, web services support the interaction between a service provider and a service requester that is independent of platforms and programming languages. The web services infrastructure includes these basic elements:

Simple Object Access Protocol (SOAP)
SOAP uses XML messages for exchanging information between service providers and service requesters. SOAP defines components of web services, which include XML messages, data types that applications use, and remote procedure calls and responses.

Web Services Description Language (WSDL)
WSDL describes what a web service can do, where the service resides, and how to invoke the service. WSDL specifies an XML vocabulary that contains all information that is needed for integration and that automates communication between web services applications.

Universal Description, Discovery, and Integration (UDDI)
UDDI provides a registry of business information, analogous to a telephone directory, that users and applications use to find required web services.

Representational State Transfer (REST)
You can use REST with the IBM Data Studio Developer tooling for Data Web Services. If you use REST bindings, you can invoke your web services with the following methods:

- HTTP GET
- HTTP POST
- HTTP POST in XML

You can use WebSphere products to build web service applications. WebSphere Studio provides tools for creating web services that include WSDL interfaces and publishing directly to a UDDI registry.

Chapter 9. Db2 sample tables

Much of the Db2 information refers to or relies on the Db2 sample tables. As a group, the tables include information that describes employees, departments, projects, and activities, and they make up a sample application that exemplifies many of the features of Db2.

GUPI

The sample storage group, databases, table spaces, tables, and views are created when you run the installation sample jobs DSNTEJ1 and DSNTEJ7. Db2 sample objects that include LOBs are created in job DSNTEJ7. All other sample objects are created in job DSNTEJ1. The CREATE INDEX statements for the sample tables are not shown here; they, too, are created by the DSNTEJ1 and DSNTEJ7 sample jobs.

Authorization on all sample objects is given to PUBLIC in order to make the sample programs easier to run. You can review the contents of any table by executing an SQL statement, for example SELECT * FROM DSN8B10.PROJ. For convenience in interpreting the examples, the department and employee tables are listed in full.

GUPI

Related concepts

Phase 1: Creating and loading sample tables (Db2 Installation and Migration)

Activity table (DSN8B10.ACT)

The activity table describes the activities that can be performed during a project.

GUPI

The activity table resides in database DSN8D11A and is created with the following statement:

```
CREATE TABLE DSN8B10.ACT
      (ACTNO    SMALLINT       NOT NULL,
       ACTKWD   CHAR(6)        NOT NULL,
       ACTDESC  VARCHAR(20)    NOT NULL,
       PRIMARY KEY (ACTNO)             )
  IN DSN8D11A.DSN8S11P
  CCSID EBCDIC;
```

GUPI

Content of the activity table

The following table shows the content of the columns in the activity table.

Table 36. Columns of the activity table

Column	Column name	Description
1	ACTNO	Activity ID (the primary key)
2	ACTKWD	Activity keyword (up to six characters)
3	ACTDESC	Activity description

The activity table has the following indexes.

Table 37. Indexes of the activity table

Name	On column	Type of index
DSN8B10.XACT1	ACTNO	Primary, ascending

© Copyright IBM Corp. 2001, 2020

Table 37. Indexes of the activity table (continued)

Name	On column	Type of index
DSN8B10.XACT2	ACTKWD	Unique, ascending

Relationship to other tables

The activity table is a parent table of the project activity table, through a foreign key on column ACTNO.

Department table (DSN8B10.DEPT)

The department table describes each department in the enterprise and identifies its manager and the department to which it reports.

GUPI

The department table resides in table space DSN8D11A.DSN8S11D and is created with the following statement:

```
CREATE TABLE DSN8B10.DEPT
      (DEPTNO    CHAR(3)           NOT NULL,
       DEPTNAME  VARCHAR(36)       NOT NULL,
       MGRNO     CHAR(6)                   ,
       ADMRDEPT  CHAR(3)           NOT NULL,
       LOCATION  CHAR(16)                  ,
       PRIMARY KEY (DEPTNO)                )
  IN DSN8D11A.DSN8S11D
  CCSID EBCDIC;
```

Because the department table is self-referencing, and also is part of a cycle of dependencies, its foreign keys must be added later with the following statements:

```
ALTER TABLE DSN8B10.DEPT
      FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8B10.DEPT
              ON DELETE CASCADE;

ALTER TABLE DSN8B10.DEPT
      FOREIGN KEY RDE (MGRNO) REFERENCES DSN8B10.EMP
              ON DELETE SET NULL;
```

GUPI

Content of the department table

The following table shows the content of the columns in the department table.

Table 38. Columns of the department table

Column	Column name	Description
1	DEPTNO	Department ID, the primary key.
2	DEPTNAME	A name that describes the general activities of the department.
3	MGRNO	Employee number (EMPNO) of the department manager.
4	ADMRDEPT	ID of the department to which this department reports; the department at the highest level reports to itself.
5	LOCATION	The remote location name.

The following table shows the indexes of the department table.

Table 39. Indexes of the department table

Name	On column	Type of index
DSN8B10.XDEPT1	DEPTNO	Primary, ascending
DSN8B10.XDEPT2	MGRNO	Ascending
DSN8B10.XDEPT3	ADMRDEPT	Ascending

The following table shows the content of the department table.

Table 40. DSN8B10.DEPT: department table

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	----------
B01	PLANNING	000020	A00	----------
C01	INFORMATION CENTER	000030	A00	----------
D01	DEVELOPMENT CENTER	------	A00	----------
E01	SUPPORT SERVICES	000050	A00	----------
D11	MANUFACTURING SYSTEMS	000060	D01	----------
D21	ADMINISTRATION SYSTEMS	000070	D01	----------
E11	OPERATIONS	000090	E01	----------
E21	SOFTWARE SUPPORT	000100	E01	----------
F22	BRANCH OFFICE F2	------	E01	----------
G22	BRANCH OFFICE G2	------	E01	----------
H22	BRANCH OFFICE H2	------	E01	----------
I22	BRANCH OFFICE I2	------	E01	----------
J22	BRANCH OFFICE J2	------	E01	----------

The LOCATION column contains null values until sample job DSNTEJ6 updates this column with the location name.

Relationship to other tables

The department table is self-referencing: the value of the administering department must be a valid department ID.

The department table is a parent table of the following :

- The employee table, through a foreign key on column WORKDEPT
- The project table, through a foreign key on column DEPTNO

The department table is a dependent of the employee table, through its foreign key on column MGRNO.

Employee table (DSN8B10.EMP)

The sample employee table identifies all employees by an employee number and lists basic personnel information.

GUPI The employee table resides in the partitioned table space DSN8D11A.DSN8S11E. Because this table has a foreign key that references DEPT, that table and the index on its primary key must be created first. Then EMP is created with the following statement:

```
CREATE TABLE DSN8B10.EMP
      (EMPNO     CHAR(6)                          NOT NULL,
       FIRSTNME  VARCHAR(12)                      NOT NULL,
       MIDINIT   CHAR(1)                          NOT NULL,
       LASTNAME  VARCHAR(15)                      NOT NULL,
       WORKDEPT  CHAR(3)                                   ,
       PHONENO   CHAR(4)         CONSTRAINT NUMBER CHECK
                 (PHONENO >= '0000' AND
                  PHONENO <= '9999')                       ,
       HIREDATE  DATE                                      ,
       JOB       CHAR(8)                                   ,
       EDLEVEL   SMALLINT                                  ,
       SEX       CHAR(1)                                   ,
       BIRTHDATE DATE                                      ,
       SALARY    DECIMAL(9,2)                              ,
       BONUS     DECIMAL(9,2)                              ,
       COMM      DECIMAL(9,2)                              ,
       PRIMARY KEY (EMPNO)                                 ,
       FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8B10.DEPT
                 ON DELETE SET NULL                        )
  EDITPROC  DSN8EAE1
  IN DSN8D11A.DSN8S11E
  CCSID EBCDIC;
```

GUPI

Content of the employee table

The following table shows the type of content of each of the columns in the employee table. The table has a check constraint, NUMBER, which checks that the four-digit phone number is in the numeric range 0000 to 9999.

Table 41. Columns of the employee table

Column	Column name	Description
1	EMPNO	Employee number (the primary key)
2	FIRSTNME	First name of employee
3	MIDINIT	Middle initial of employee
4	LASTNAME	Last name of employee
5	WORKDEPT	ID of department in which the employee works
6	PHONENO	Employee telephone number
7	HIREDATE	Date of hire
8	JOB	Job held by the employee
9	EDLEVEL	Number of years of formal education
10	SEX	Sex of the employee (M or F)
11	BIRTHDATE	Date of birth
12	SALARY	Yearly salary in dollars
13	BONUS	Yearly bonus in dollars

Table 41. Columns of the employee table (continued)

Column	Column name	Description
14	COMM	Yearly commission in dollars

The following table shows the indexes of the employee table.

Table 42. Indexes of the employee table

Name	On column	Type of index
DSN8B10.XEMP1	EMPNO	Primary, partitioned, ascending
DSN8B10.XEMP2	WORKDEPT	Ascending

The following table shows the first half (left side) of the content of the employee table. (Table 44 on page 324 shows the remaining content (right side) of the employee table.)

Table 43. Left half of DSN8B10.EMP: employee table. Note that a blank in the MIDINIT column is an actual value of " " rather than null.

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10
000030	SALLY	A	KWAN	C01	4738	1975-04-05
000050	JOHN	B	GEYER	E01	6789	1949-08-17
000060	IRVING	F	STERN	D11	6423	1973-09-14
000070	EVA	D	PULASKI	D21	7831	1980-09-30
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16
000120	SEAN		O'CONNELL	A00	2167	1963-12-05
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15
000150	BRUCE		ADAMSON	D11	4510	1972-02-12
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07
000190	JAMES	H	WALKER	D11	2986	1974-07-26
000200	DAVID		BROWN	D11	4501	1966-03-03
000210	WILLIAM	T	JONES	D11	0942	1979-04-11
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1969-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11

Table 43. Left half of DSN8B10.EMP: employee table. Note that a blank in the MIDINIT column is an actual value of " " rather than null. *(continued)*

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5698	1947-05-05
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01
200120	GREG		ORLANDO	A00	2167	1972-05-05
200140	KIM	N	NATZ	C01	1793	1976-12-15
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15
200220	REBA	K	JOHN	D11	0672	1968-08-29
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12
200330	HELENA		WONG	E21	2103	1976-02-23
200340	ROY	R	ALONZO	E21	5698	1947-05-05

(Table 43 on page 323 shows the first half (right side) of the content of employee table.)

Table 44. Right half of DSN8B10.EMP: employee table

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000010)	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
(000020)	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
(000030)	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
(000050)	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
(000060)	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
(000070)	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
(000090)	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
(000100)	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
(000110)	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
(000120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(000130)	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
(000140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(000150)	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
(000160)	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00

Table 44. Right half of DSN8B10.EMP: employee table (continued)

(EMPNO)	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
(000170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(000180)	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
(000190)	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
(000200)	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
(000210)	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
(000220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(000230)	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
(000240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(000250)	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
(000260)	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
(000270)	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
(000280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(000290)	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
(000300)	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00
(000310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
(000320)	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
(000330)	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
(000340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00
(200010)	SALESREP	18	F	1933-08-14	46500.00	1000.00	4220.00
(200120)	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
(200140)	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
(200170)	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
(200220)	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
(200240)	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
(200280)	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
(200310)	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
(200330)	FIELDREP	14	F	1941-07-18	25370.00	500.00	2030.00
(200340)	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

Relationship to other tables

The employee table is a parent table of:

- The department table, through a foreign key on column MGRNO
- The project table, through a foreign key on column RESPEMP

The employee table is a dependent of the department table, through its foreign key on column WORKDEPT.

Employee photo and resume table (DSN8B10.EMP_PHOTO_RESUME)

The sample employee photo and resume table complements the employee table.

GUPI Each row of the photo and resume table contains a photo of the employee, in two formats, and the employee's resume. The photo and resume table resides in table space .DSN8S11B. The following statement creates the table:

```
CREATE TABLE  DSN8B10.EMP_PHOTO_RESUME
               (EMPNO      CHAR(06) NOT NULL,
                EMP_ROWID  ROWID NOT NULL GENERATED ALWAYS,
                PSEG_PHOTO BLOB(500K),
                BMP_PHOTO  BLOB(100K),
                RESUME        CLOB(5K))
                PRIMARY KEY  (EMPNO)
       IN  DSN8D11L.DSN8S11B
       CCSID  EBCDIC;
```

Db2 requires an auxiliary table for each LOB column in a table. The following statements define the auxiliary tables for the three LOB columns in DSN8B10.EMP_PHOTO_RESUME:

```
CREATE AUX TABLE DSN8B10.AUX_BMP_PHOTO
               IN DSN8D11L.DSN8S11M
               STORES DSN8B10.EMP_PHOTO_RESUME
               COLUMN BMP_PHOTO;

CREATE AUX TABLE DSN8B10.AUX_PSEG_PHOTO
               IN DSN8D11L.DSN8S11L
               STORES DSN8B10.EMP_PHOTO_RESUME
               COLUMN PSEG_PHOTO;

CREATE AUX TABLE DSN8B10.AUX_EMP_RESUME
               IN DSN8D11L.DSN8S11N
               STORES DSN8B10.EMP_PHOTO_RESUME
               COLUMN RESUME;
```

GUPI

Content of the employee photo and resume table

The following table shows the content of the columns in the employee photo and resume table.

Table 45. Columns of the employee photo and resume table

Column	Column name	Description
1	EMPNO	Employee ID (the primary key).
2	EMP_ROWID	Row ID to uniquely identify each row of the table. Db2 supplies the values of this column.
3	PSEG_PHOTO	Employee photo, in PSEG format.
4	BMP_PHOTO	Employee photo, in BMP format.
5	RESUME	Employee resume.

The following table shows the indexes for the employee photo and resume table.

Table 46. Indexes of the employee photo and resume table

Name	On column	Type of index
DSN8B10.XEMP_PHOTO_RESUME	EMPNO	Primary, ascending

The following table shows the indexes for the auxiliary tables that support the employee photo and resume table.

Table 47. Indexes of the auxiliary tables for the employee photo and resume table

Name	On table	Type of index
DSN8B10.XAUX_BMP_PHOTO	DSN8B10.AUX_BMP_PHOTO	Unique
DSN8B10.XAUX_PSEG_PHOTO	DSN8B10.AUX_PSEG_PHOTO	Unique
DSN8B10.XAUX_EMP_RESUME	DSN8B10.AUX_EMP_RESUME	Unique

Relationship to other tables

The employee photo and resume table is a parent table of the project table, through a foreign key on column RESPEMP.

Project table (DSN8B10.PROJ)

The sample project table describes each project that the business is currently undertaking. Data that is contained in each row of the table includes the project number, name, person responsible, and schedule dates.

The project table resides in database DSN8D11A. Because this table has foreign keys that reference DEPT and EMP, those tables and the indexes on their primary keys must be created first. Then PROJ is created with the following statement:

GUPI

```
CREATE TABLE DSN8B10.PROJ
               (PROJNO   CHAR(6) PRIMARY KEY NOT NULL,
                PROJNAME VARCHAR(24)    NOT NULL WITH DEFAULT
                  'PROJECT NAME UNDEFINED',
                DEPTNO   CHAR(3)        NOT NULL REFERENCES
                  DSN8B10.DEPT ON DELETE RESTRICT,
                RESPEMP  CHAR(6)        NOT NULL REFERENCES
                  DSN8B10.EMP ON DELETE RESTRICT,
                PRSTAFF  DECIMAL(5, 2)          ,
                PRSTDATE DATE                   ,
                PRENDATE DATE                   ,
                MAJPROJ  CHAR(6))
        IN DSN8D11A.DSN8S11P
        CCSID EBCDIC;
```

Because the project table is self-referencing, the foreign key for that constraint must be added later with the following statement:

```
ALTER TABLE DSN8B10.PROJ
      FOREIGN KEY RPP (MAJPROJ) REFERENCES DSN8B10.PROJ
              ON DELETE CASCADE;
```

GUPI

Content of the project table

The following table shows the content of the columns of the project table.

Table 48. Columns of the project table

Column	Column name	Description
1	PROJNO	Project ID (the primary key)
2	PROJNAME	Project name
3	DEPTNO	ID of department responsible for the project
4	RESPEMP	ID of employee responsible for the project

Table 48. Columns of the project table (continued)

Column	Column name	Description
5	PRSTAFF	Estimated mean number of persons that are needed between PRSTDATE and PRENDATE to complete the whole project, including any subprojects
6	PRSTDATE	Estimated project start date
7	PRENDATE	Estimated project end date
8	MAJPROJ	ID of any project of which this project is a part

The following table shows the indexes for the project table:

Table 49. Indexes of the project table

Name	On column	Type of index
DSN8B10.XPROJ1	PROJNO	Primary, ascending
DSN8B10.XPROJ2	RESPEMP	Ascending

Relationship to other tables

The table is self-referencing: a non-null value of MAJPROJ must be a valid project number. The table is a parent table of the project activity table, through a foreign key on column PROJNO. It is a dependent of the following tables:

- The department table, through its foreign key on DEPTNO
- The employee table, through its foreign key on RESPEMP

Project activity table (DSN8B10.PROJACT)

The sample project activity table lists the activities that are performed for each project.

The project activity table resides in database DSN8D11A. Because this table has foreign keys that reference PROJ and ACT, those tables and the indexes on their primary keys must be created first. Then PROJACT is created with the following statement:

GUPI

```
CREATE TABLE DSN8B10.PROJACT
      (PROJNO    CHAR(6)                        NOT NULL,
       ACTNO     SMALLINT                       NOT NULL,
       ACSTAFF   DECIMAL(5,2)                           ,
       ACSTDATE  DATE                           NOT NULL,
       ACENDATE  DATE                                   ,
       PRIMARY KEY (PROJNO, ACTNO, ACSTDATE),
       FOREIGN KEY RPAP (PROJNO) REFERENCES DSN8B10.PROJ
                                    ON DELETE RESTRICT,
       FOREIGN KEY RPAA (ACTNO) REFERENCES DSN8B10.ACT
                                    ON DELETE RESTRICT)
  IN DSN8D11A.DSN8S11P
  CCSID EBCDIC;
```

GUPI

Content of the project activity table

The following table shows the content of the columns of the project activity table.

Table 50. Columns of the project activity table

Column	Column name	Description
1	PROJNO	Project ID
2	ACTNO	Activity ID
3	ACSTAFF	Estimated mean number of employees that are needed to staff the activity
4	ACSTDATE	Estimated activity start date
5	ACENDATE	Estimated activity completion date

The following table shows the index of the project activity table:

Table 51. Index of the project activity table

Name	On columns	Type of index
DSN8B10.XPROJAC1	PROJNO, ACTNO, ACSTDATE	primary, ascending

Relationship to other tables

The project activity table is a parent table of the employee to project activity table, through a foreign key on columns PROJNO, ACTNO, and EMSTDATE. It is a dependent of the following tables:

- The activity table, through its foreign key on column ACTNO
- The project table, through its foreign key on column PROJNO

Related reference

Activity table (DSN8B10.ACT)
The activity table describes the activities that can be performed during a project.

Project table (DSN8B10.PROJ)
The sample project table describes each project that the business is currently undertaking. Data that is contained in each row of the table includes the project number, name, person responsible, and schedule dates.

Employee-to-project activity table (DSN8B10.EMPPROJACT)

The sample employee-to-project activity table identifies the employee who performs an activity for a project, tells the proportion of the employee's time that is required, and gives a schedule for the activity.

GUPI

The employee-to-project activity table resides in database DSN8D11A. Because this table has foreign keys that reference EMP and PROJACT, those tables and the indexes on their primary keys must be created first. Then EMPPROJACT is created with the following statement:

```
CREATE TABLE DSN8B10.EMPPROJACT
      (EMPNO     CHAR(6)                        NOT NULL,
       PROJNO    CHAR(6)                        NOT NULL,
       ACTNO     SMALLINT                       NOT NULL,
       EMPTIME   DECIMAL(5,2)                           ,
       EMSTDATE  DATE                                   ,
       EMENDATE  DATE                                   ,
       FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
                 REFERENCES DSN8B10.PROJACT
                                    ON DELETE RESTRICT,
       FOREIGN KEY REPAE (EMPNO) REFERENCES DSN8B10.EMP
                                    ON DELETE RESTRICT)
  IN DSN8D11A.DSN8S11P
  CCSID EBCDIC;
```

GUPI

Content of the employee-to-project activity table

The following table shows the content of the columns in the employee-to-project activity table.

Table 52. Columns of the employee-to-project activity table

Column	Column name	Description
1	EMPNO	Employee ID number
2	PROJNO	Project ID of the project
3	ACTNO	ID of the activity within the project
4	EMPTIME	A proportion of the employee's full time (between 0.00 and 1.00) that is to be spent on the activity
5	EMSTDATE	Date the activity starts
6	EMENDATE	Date the activity ends

The following table shows the indexes for the employee-to-project activity table:

Table 53. Indexes of the employee-to-project activity table

Name	On columns	Type of index
DSN8B10.XEMPPROJACT1	PROJNO, ACTNO, EMSTDATE, EMPNO	Unique, ascending
DSN8B10.XEMPPROJACT2	EMPNO	Ascending

Relationship to other tables

The employee-to-project activity table is a dependent of the following tables:

- The employee table, through its foreign key on column EMPNO
- The project activity table, through its foreign key on columns PROJNO, ACTNO, and EMSTDATE.

Related reference

Employee table (DSN8B10.EMP)
The sample employee table identifies all employees by an employee number and lists basic personnel information.

Project activity table (DSN8B10.PROJACT)
The sample project activity table lists the activities that are performed for each project.

Unicode sample table (DSN8B10.DEMO_UNICODE)

The Unicode sample table is used to verify that data conversions to and from EBCDIC and Unicode are working as expected.

GUPI

The table resides in database DSN8D11A, and is defined with the following statement:

```
CREATE TABLE DSN8B10.DEMO_UNICODE
           (LOWER_A_TO_Z      CHAR(26)                    ,
            UPPER_A_TO_Z      CHAR(26)                    ,
            ZERO_TO_NINE      CHAR(10)                    ,
            X00_TO_XFF    VARCHAR(256)        FOR BIT DATA)
   IN DSN8D81E.DSN8S81U
   CCSID UNICODE;
```

GUPI

Content of the Unicode sample table

The following table shows the content of the columns in the Unicode sample table:

Table 54. Columns of the Unicode sample table

Column	Column Name	Description
1	LOWER_A_TO_Z	Array of characters, 'a' to 'z'
2	UPPER_A_TO_Z	Array of characters, 'A' to 'Z'
3	ZERO_TO_NINE	Array of characters, '0' to '9'
4	X00_TO_XFF	Array of characters, x'00' to x'FF'

This table has no indexes.

Relationship to other tables

This table has no relationship to other tables.

Relationships among the sample tables

Relationships among the sample tables are established by foreign keys in dependent tables that reference primary keys in parent tables.

The following figure shows relationships among the sample tables. You can find descriptions of the columns with the descriptions of the tables.

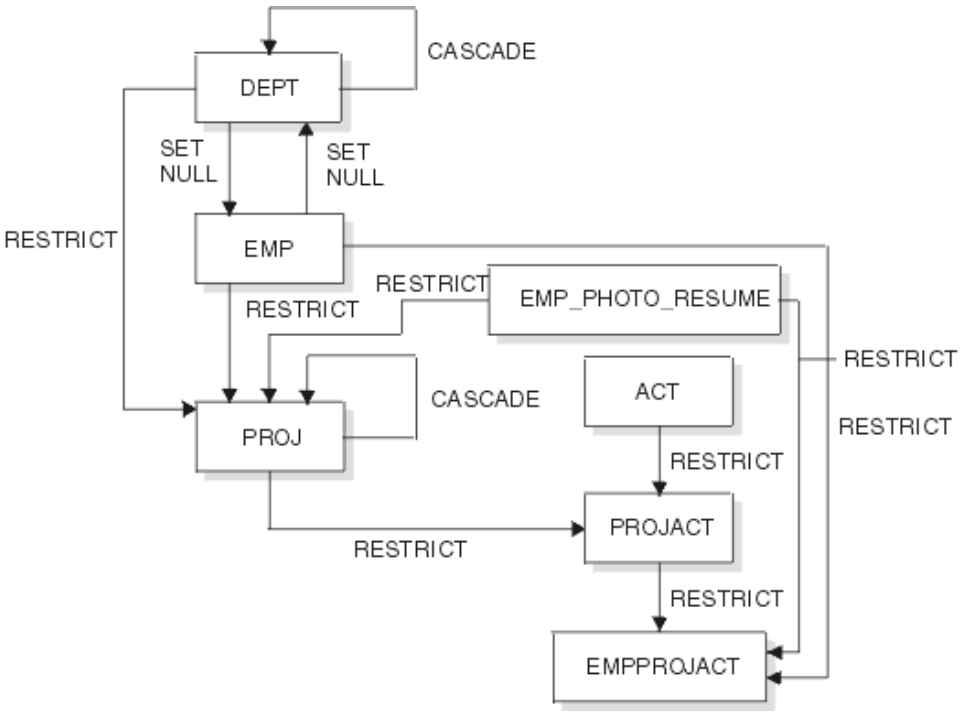


Figure 70. Relationships among tables in the sample application

Related reference

Activity table (DSN8B10.ACT)
The activity table describes the activities that can be performed during a project.

Department table (DSN8B10.DEPT)

The department table describes each department in the enterprise and identifies its manager and the department to which it reports.

Employee photo and resume table (DSN8B10.EMP_PHOTO_RESUME)
The sample employee photo and resume table complements the employee table.

Employee table (DSN8B10.EMP)
The sample employee table identifies all employees by an employee number and lists basic personnel information.

Employee-to-project activity table (DSN8B10.EMPPROJACT)
The sample employee-to-project activity table identifies the employee who performs an activity for a project, tells the proportion of the employee's time that is required, and gives a schedule for the activity.

Project activity table (DSN8B10.PROJACT)
The sample project activity table lists the activities that are performed for each project.

Project table (DSN8B10.PROJ)
The sample project table describes each project that the business is currently undertaking. Data that is contained in each row of the table includes the project number, name, person responsible, and schedule dates.

Unicode sample table (DSN8B10.DEMO_UNICODE)
The Unicode sample table is used to verify that data conversions to and from EBCDIC and Unicode are working as expected.

Views on the sample tables

Db2 creates a number of views on the sample tables for use in the sample applications.

The following table indicates the tables on which each view is defined and the sample applications that use the view. All view names have the qualifier DSN8B10.

Table 55. Views on sample tables

View name	On tables or views	Used in application
VDEPT	DEPT	Organization Project
VHDEPT	DEPT	Distributed organization
VEMP	EMP	Distributed organization Organization Project
VPROJ	PROJ	Project
VACT	ACT	Project
VPROJACT	PROJACT	Project
VEMPPROJACT	EMPPROJACT	Project
VDEPMG1	DEPT EMP	Organization
VEMPDPT1	DEPT EMP	Organization
VASTRDE1	DEPT	

Table 55. Views on sample tables (continued)

View name	On tables or views	Used in application
VASTRDE2	VDEPMG1 EMP	Organization
VPROJRE1	PROJ EMP	Project
VPSTRDE1	VPROJRE1 VPROJRE2	Project
VPSTRDE2	VPROJRE1	Project
VFORPLA	VPROJRE1 EMPPROJACT	Project
VSTAFAC1	PROJACT ACT	Project
VSTAFAC2	EMPPROJACT ACT EMP	Project
VPHONE	EMP DEPT	Phone
VEMPLP	EMP	Phone

GUPI

The following SQL statement creates the view named VDEPT.

```
CREATE VIEW DSN8B10.VDEPT
   AS SELECT ALL      DEPTNO  ,
                      DEPTNAME,
                      MGRNO   ,
                      ADMRDEPT
   FROM DSN8B10.DEPT;
```

The following SQL statement creates the view named VHDEPT.

```
 CREATE VIEW DSN8B10.VHDEPT
    AS SELECT ALL      DEPTNO  ,
                       DEPTNAME,
                       MGRNO   ,
                       ADMRDEPT,
                       LOCATION
    FROM DSN8B10.DEPT;
```

The following SQL statement creates the view named VEMP.

```
CREATE VIEW DSN8B10.VEMP
   AS SELECT ALL      EMPNO   ,
                      FIRSTNME,
                      MIDINIT ,
                      LASTNAME,
                      WORKDEPT
   FROM DSN8B10.EMP;
```

The following SQL statement creates the view named VPROJ.

```
CREATE VIEW DSN8B10.VPROJ
   AS SELECT ALL
             PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF,
             PRSTDATE, PRENDATE, MAJPROJ
   FROM DSN8B10.PROJ ;
```

The following SQL statement creates the view named VACT.

```
CREATE VIEW DSN8B10.VACT
      AS SELECT ALL   ACTNO   ,
                      ACTKWD  ,
                      ACTDESC
           FROM DSN8B10.ACT ;
```

The following SQL statement creates the view named VPROJACT.

```
CREATE VIEW DSN8B10.VPROJACT
      AS SELECT ALL
            PROJNO,ACTNO, ACSTAFF, ACSTDATE, ACENDATE
            FROM DSN8B10.PROJACT ;
```

The following SQL statement creates the view named VEMPPROJACT.

```
CREATE VIEW DSN8B10.VEMPPROJACT
      AS SELECT ALL
          EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, EMENDATE
          FROM DSN8B10.EMPPROJACT ;
```

The following SQL statement creates the view named VDEPMG1.

```
CREATE VIEW DSN8B10.VDEPMG1
      (DEPTNO, DEPTNAME, MGRNO, FIRSTNME, MIDINIT,
       LASTNAME, ADMRDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, FIRSTNME, MIDINIT,
        LASTNAME, ADMRDEPT
        FROM DSN8B10.DEPT LEFT OUTER JOIN DSN8B10.EMP
        ON MGRNO = EMPNO ;
```

The following SQL statement creates the view named VEMPDPT1.

```
CREATE VIEW DSN8B10.VEMPDPT1
      (DEPTNO, DEPTNAME, EMPNO, FRSTINIT, MIDINIT,
       LASTNAME, WORKDEPT)
    AS SELECT ALL
       DEPTNO, DEPTNAME, EMPNO, SUBSTR(FIRSTNME, 1, 1), MIDINIT,
       LASTNAME, WORKDEPT
        FROM DSN8B10.DEPT RIGHT OUTER JOIN DSN8B10.EMP
        ON WORKDEPT = DEPTNO ;
```

The following SQL statement creates the view named VASTRDE1.

```
CREATE VIEW DSN8B10.VASTRDE1
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
        D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
        D1.LASTNAME, '1',
        D2.DEPTNO,D2.DEPTNAME,D2.MGRNO,D2.FIRSTNME,D2.MIDINIT,
        D2.LASTNAME
        FROM DSN8B10.VDEPMG1 D1, DSN8B10.VDEPMG1 D2
        WHERE D1.DEPTNO = D2.ADMRDEPT ;
```

The following SQL statement creates the view named VASTRDE2.

```
CREATE VIEW DSN8B10.VASTRDE2
    (DEPT1NO,DEPT1NAM,EMP1NO,EMP1FN,EMP1MI,EMP1LN,TYPE2,
     DEPT2NO,DEPT2NAM,EMP2NO,EMP2FN,EMP2MI,EMP2LN)
    AS SELECT ALL
         D1.DEPTNO,D1.DEPTNAME,D1.MGRNO,D1.FIRSTNME,D1.MIDINIT,
```

```
         D1.LASTNAME,'2',
         D1.DEPTNO,D1.DEPTNAME,E2.EMPNO,E2.FIRSTNME,E2.MIDINIT,
         E2.LASTNAME
         FROM DSN8B10.VDEPMG1 D1, DSN8B10.EMP E2
         WHERE D1.DEPTNO = E2.WORKDEPT;
```

The following figure shows the SQL statement that creates the view named VPROJRE1.

```
CREATE VIEW DSN8B10.VPROJRE1
  (PROJNO,PROJNAME,PROJDEP,RESPEMP,FIRSTNME,MIDINIT,
   LASTNAME,MAJPROJ)
   AS SELECT ALL
      PROJNO,PROJNAME,DEPTNO,EMPNO,FIRSTNME,MIDINIT,
      LASTNAME,MAJPROJ
     FROM DSN8B10.PROJ, DSN8B10.EMP
     WHERE RESPEMP = EMPNO ;
```

Figure 71. VPROJRE1

The following SQL statement creates the view named VPSTRDE1.

```
CREATE VIEW DSN8B10.VPSTRDE1
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P2.PROJNO,P2.PROJNAME,P2.RESPEMP,P2.FIRSTNME,P2.MIDINIT,
        P2.LASTNAME
     FROM DSN8B10.VPROJRE1 P1,
       DSN8B10.VPROJRE1 P2
     WHERE P1.PROJNO = P2.MAJPROJ ;
```

The following SQL statement creates the view named VPSTRDE2.

```
CREATE VIEW DSN8B10.VPSTRDE2
  (PROJ1NO,PROJ1NAME,RESP1NO,RESP1FN,RESP1MI,RESP1LN,
   PROJ2NO,PROJ2NAME,RESP2NO,RESP2FN,RESP2MI,RESP2LN)
   AS SELECT ALL
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME,
        P1.PROJNO,P1.PROJNAME,P1.RESPEMP,P1.FIRSTNME,P1.MIDINIT,
        P1.LASTNAME
     FROM DSN8B10.VPROJRE1 P1
       WHERE NOT EXISTS
         (SELECT * FROM DSN8B10.VPROJRE1 P2
           WHERE P1.PROJNO = P2.MAJPROJ) ;
```

The following SQL statement creates the view named VFORPLA.

```
CREATE VIEW DSN8B10.VFORPLA
  (PROJNO,PROJNAME,RESPEMP,PROJDEP,FRSTINIT,MIDINIT,LASTNAME)
   AS SELECT ALL
      F1.PROJNO,PROJNAME,RESPEMP,PROJDEP, SUBSTR(FIRSTNME, 1, 1),
      MIDINIT, LASTNAME
     FROM DSN8B10.VPROJRE1 F1 LEFT OUTER JOIN DSN8B10.EMPPROJACT F2
     ON F1.PROJNO = F2.PROJNO;
```

The following SQL statement creates the view named VSTAFAC1.

```
CREATE VIEW DSN8B10.VSTAFAC1
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
   EMPTIME,STDATE,ENDATE, TYPE)
   AS SELECT ALL
        PA.PROJNO, PA.ACTNO, AC.ACTDESC,' ', ' ', ' ', ' ',
        PA.ACSTAFF, PA.ACSTDATE,
        PA.ACENDATE,'1'
     FROM DSN8B10.PROJACT PA, DSN8B10.ACT AC
     WHERE PA.ACTNO = AC.ACTNO ;
```

The following SQL statement creates the view named VSTAFAC2.

```
CREATE VIEW DSN8B10.VSTAFAC2
  (PROJNO, ACTNO, ACTDESC, EMPNO, FIRSTNME, MIDINIT, LASTNAME,
```

```
   EMPTIME,STDATE, ENDATE, TYPE)
   AS SELECT ALL
         EP.PROJNO, EP.ACTNO, AC.ACTDESC, EP.EMPNO,EM.FIRSTNME,
         EM.MIDINIT, EM.LASTNAME, EP.EMPTIME, EP.EMSTDATE,
         EP.EMENDATE,'2'
     FROM DSN8B10.EMPPROJACT EP, DSN8B10.ACT AC, DSN8B10.EMP EM
     WHERE EP.ACTNO = AC.ACTNO  AND EP.EMPNO = EM.EMPNO ;
```

The following SQL statement creates the view named VPHONE.

```
CREATE VIEW DSN8B10.VPHONE
              (LASTNAME,
               FIRSTNAME,
               MIDDLEINITIAL,
               PHONENUMBER,
               EMPLOYEENUMBER,
               DEPTNUMBER,
               DEPTNAME)
   AS SELECT ALL      LASTNAME,
                      FIRSTNME,
                      MIDINIT ,
                      VALUE(PHONENO,'    '),
                      EMPNO,
                      DEPTNO,
                      DEPTNAME
   FROM DSN8B10.EMP, DSN8B10.DEPT
   WHERE WORKDEPT = DEPTNO;
```

The following SQL statement creates the view named VEMPLP.

```
CREATE VIEW DSN8B10.VEMPLP
              (EMPLOYEENUMBER,
               PHONENUMBER)
   AS SELECT ALL      EMPNO   ,
                      PHONENO
   FROM DSN8B10.EMP ;
```

GUPI

Storage of sample application tables

Normally, related data is stored in the same database.

The following figure shows how the sample tables are related to databases and storage groups. Two databases are used to illustrate the possibility.

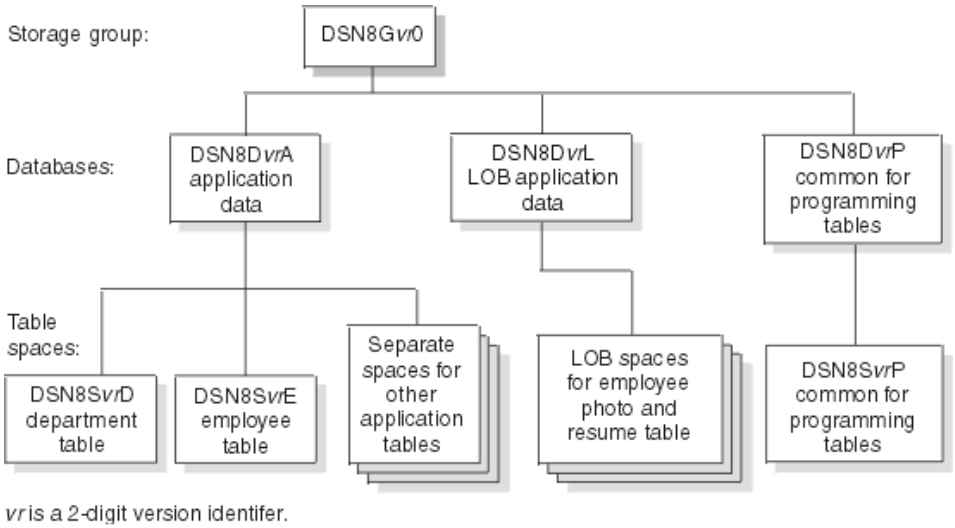


Figure 72. Relationship among sample databases and table spaces

In addition to the storage group and databases that are shown in the preceding figure, the storage group DSN8G11U and database DSN8D11U are created when you run DSNTEJ2A.

Storage group for sample application data

Sample application data is stored in storage group DSN8G110. The default storage group, SYSDEFLT, which is created when Db2 is installed, is not used to store sample application data.

GUPI

The storage group that is used to store sample application data is defined by the following statement:

```
CREATE STOGROUP DSN8G110
  VOLUMES (DSNV01)
  VCAT  DSNC110;
```

GUPI

Databases for sample application data

Sample application data is stored in several different databases. The default database that is created when Db2 is installed is not used to store the sample application data.

GUPI DSN8D11P is the database that is used for tables that are related to programs. The other databases are used for tables that are related to applications. The databases are defined by the following statements:

```
CREATE DATABASE DSN8D11A
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D11P
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D11L
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID EBCDIC;

CREATE DATABASE DSN8D11E
  STOGROUP DSN8G110
  BUFFERPOOL BP0
  CCSID UNICODE;

CREATE DATABASE DSN8D11U
  STOGROUP DSN8G11U
  CCSID EBCDIC;
```

GUPI

Table spaces for sample application data

The table spaces that are not explicitly defined are created implicitly in the DSN8D11A database, using the default space attributes.

GUPI

The following SQL statements explicitly define a series of table spaces.

```
CREATE TABLESPACE DSN8S11D
  IN DSN8D11A
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S11E
  IN DSN8D11A
```

```
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  NUMPARTS 4
    (PART 1 USING STOGROUP DSN8G110
                  PRIQTY 12
                  SECQTY 12,
     PART 3 USING STOGROUP DSN8G110
                  PRIQTY 12
                  SECQTY 12)
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  COMPRESS YES
  CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S11B
  IN DSN8D11L
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE
  LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
```

```
CREATE LOB TABLESPACE DSN8S11M
  IN DSN8D11L
  LOG NO;

CREATE LOB TABLESPACE DSN8S11L
  IN DSN8D11L
  LOG NO;

CREATE LOB TABLESPACE DSN8S11N
  IN DSN8D11L
  LOG NO;
```

```
CREATE TABLESPACE DSN8S11C
  IN DSN8D11P
  USING STOGROUP DSN8G110
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE TABLE
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;

CREATE TABLESPACE DSN8S11P
  IN DSN8D11A
  USING STOGROUP DSN8G110
        PRIQTY 160
        SECQTY 80
  SEGSIZE 4
  LOCKSIZE ROW
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S11R
  IN DSN8D11A
  USING STOGROUP DSN8G110
        PRIQTY 20
        SECQTY 20
        ERASE NO
  LOCKSIZE PAGE LOCKMAX SYSTEM
  BUFFERPOOL BP0
  CLOSE NO
  CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S11S
   IN DSN8D11A
   USING STOGROUP DSN8G110
```

```
            PRIQTY 20
            SECQTY 20
            ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S81Q
   IN DSN8D81P
   USING STOGROUP DSN8G810
            PRIQTY 160
            SECQTY 80
   SEGSIZE 4
   LOCKSIZE PAGE
   BUFFERPOOL BP0
   CLOSE NO
   CCSID EBCDIC;
```

```
CREATE TABLESPACE DSN8S81U
   IN DSN8D81E
   USING STOGROUP DSN8G810
            PRIQTY 5
            SECQTY 5
            ERASE NO
   LOCKSIZE PAGE LOCKMAX SYSTEM
   BUFFERPOOL BP0
   CLOSE NO
   CCSID UNICODE;
```

GUPI

SYSDUMMY*x* tables

Db2 for z/OS provides a set of SYSDUMMY*x* catalog tables.

GUPI

The last character of the table name identifies the associated encoding scheme as follows:

- SYSIBM.SYSDUMMY1 uses the EBCDIC encoding scheme.
- SYSIBM.SYSDUMMYE uses the EBCDIC encoding scheme.
- SYSIBM.SYSDUMMYA uses the ASCII encoding scheme.
- SYSIBM.SYSDUMMYU uses the UNICODE encoding scheme.

Although the SYSDUMMY*x* tables are implemented as catalog tables, they are similar to sample tables, and are used in some examples in the Db2 for z/OS documentation.

You can use any of the SYSDUMMY*x* tables when you need to write a query, but no data from a table is referenced. In any query, you must specify a table reference in the FROM clause, but if the query does not reference any data from the table, it does not matter which table is referenced in the FROM clause. Each of the SYSDUMMY*x* tables contains one row, so a SYSDUMMY*x* table can be referenced in a SELECT INTO statement without the need for a predicate to limit the result to a single row of data.

For example, when you want to retrieve the value of a special register or global variable, you can use a query that references a SYSDUMMY*x* table.

```
SELECT CURRENT PATH -- Retrieve the value of a special register
 INTO :myvar
 FROM SYSIBM.SYSDUMMY1;
SELECT CLIENT_IPADDR -- Retrieve the value of a special register
 INTO :myvar
 FROM SYSIBM.SYSDUMMY1;
```

Sometimes when Db2 for z/OS processes an SQL statement, the statement is rewritten, and a reference to a SYSDUMMY*x* table is added. For example, some of the internal rewrites that result in adding a reference to a SYSDUMMY*x* table are for processing the search condition of a trigger, or the SQL PL

control statements IF, REPEAT, RETURN, or WHILE. In these situations, the privilege set must include the SELECT privilege on the SYSDUMMY*x* table that is referenced.

GUPI

Related concepts

Objects with different CCSIDs in the same SQL statement (Db2 Internationalization Guide)

Related tasks

Avoiding character conversion for LOB locators (Db2 Application programming and SQL)

Related reference

SYSDUMMY1 catalog table (Db2 SQL)

SYSDUMMYA catalog table (Db2 SQL)

SYSDUMMYE catalog table (Db2 SQL)

SYSDUMMYU catalog table (Db2 SQL)

Chapter 10. Db2 basics tutorial: Working with Db2 for z/OS interactively

This tutorial shows you how to use Db2 for z/OS in the z/OS terminal interface to complete some basic database administration tasks.

You might use many different tools and environments to work with Db2 for z/OS in your job. Many of these methods use batch jobs or database applications that are programmed in advance and that run in the background, with little or no direct human interaction.

However, you can also work with Db2 for z/OS through *interactive* methods such as commands and menu-style user interfaces. For example, you can use the SPUFI (SQL processor using file input) facility within the DB2I (Db2 Interactive) primary option menu of ISPF to work with SQL statements interactively. You can edit SQL statements in SPUFI, issue the statements, and see the results immediately.

This tutorial consists of two modules:

- "Module 1: Querying and modifying data" on page 342
- "Module 2: Running Db2 utilities" on page 353

Learning objectives

- Become familiar with working with Db2 in the z/OS terminal interface.
- Learn how to query and modify data interactively by submitting SQL statements through SPUFI.
- Learn how to create Db2 tables interactively by using SPUFI.
- Learn how to run Db2 utilities from the DB2I utilities panel and by submitting JCL jobs.

Audience

This tutorial is intended for new database professionals who have some familiarity with the basic concepts of relational database management systems and the SQL language, but who are new to Db2 for z/OS and the z/OS terminal interface.

Users of this tutorial will be more successful if they are familiar with the following information:

- The basic concepts of relational database data structures, such as databases and tables
- The basic concepts of the SQL language
- How to log on to z/OS by using a TSO session
- How to work with TSO panels
- How to work with ISPF
- How to edit data sets in the ISPF editor

Before you begin

- Ensure that you have access to a supported version of Db2 and that the Db2 sample tables have been created.
- Determine the name of the Db2 subsystem that contains the sample tables. This tutorial uses the subsystem name DB2A.
- Obtain a Db2 authorization ID and password that has appropriate authorities to query the Db2 sample tables and create objects. This tutorial uses the authorization ID TUTOR01.

© Copyright IBM Corp. 2001, 2020

Conventions used in this tutorial

- This tutorial uses standard z/OS terminal key names. These keys might be mapped differently on your keyboard. For example, the standard z/OS terminal key name is PF3. In the standard mapping, the PF3 key corresponds to the F3 key on the keyboard; however, the key that you press depends on your keyboard mapping.

  For more information about keyboard mapping in TSO and ISPF, see the following topics:

  – Special keys on your keyboard in TSO/E (TSO/E Primer)
  – ISPF keyboard keys and functions (z/OS Basic Skills)
- Samples refer to the sample tables that are supplied with Db2 for z/OS.
- Certain panels, especially outside of DB2I, are likely to be customized at your site and are likely to have different names than the names that are used in this tutorial.

  For example, the "**ISPF/PDF Primary Option Menu**" is the default name of a panel that is configured during a step of the Db2 for z/OS installation process. This panel might have a different name at your site. If you cannot find the corresponding panels, ask a colleague for help.

.

Related concepts

Db2 for z/OS and SQL concepts
Many structures and processes are associated with a relational database. The structures are the key components of a Db2 database system, and the processes are the interactions that occur when applications access the database system.

Related information

TSO/E Primer

Module 1: Querying and modifying data

You can interactively query and modify data that is stored in Db2 for z/OS by issuing SQL statements through SPUFI.

Learning objectives

This tutorial shows you how to do the following tasks:

- Issue SQL statements by using SPUFI
- Query data in Db2 tables and view results
- Modify data in Db2 tables and verify the changes
- Create objects

Time required

45 minutes.

Before you begin

- Complete the prerequisites that are listed in the introduction to this tutorial.
- Log on to z/OS by using a TSO session.
- Create or obtain access to a partitioned data set to contain your work. Contact your system administrator to find out the conventions for data sets at your site, or accept the default values. This tutorial assumes that the data set is named USER.SRCLIB.TUTORIAL.

Lesson 1.1: Querying data interactively

One of the most basic database tasks is to query the data that the database contains. Even if you are new to Db2 for z/OS, you might already know how to query data in other databases. In fact, you can use many different methods and tools to access data that is stored in Db2 for z/OS databases, including tools that

provide modern graphical user-interfaces. However, this lesson provides an opportunity for you learn how to query data by using the commands and menus of the panel-based SPUFI facility in the z/OS terminal interface.

Before you begin
Before you begin

- Complete the prerequisites that are described in "Module 1: Querying and modifying data" on page 342.
- Start an ISPF session.

About this task

Procedure
Procedure

1. From the **ISPF/PDF Primary Option Menu**, type the option from the menu that includes Db2, and press Enter.

   Your site probably uses a customized version of this menu. In the following example, option D is selected.

```
   Menu  Utilities  Compilers  Options  Status  Help

                          ISPF/PDF Primary Option Menu
 Option ===> D

 0  Settings      Terminal and user parameters           User ID . : TUTOR01
 1  View          Display source data or listings        Time. . . : 11:10
 2  Edit          Create or change source data           Terminal. : 3278A
 3  Utilities     Perform utility functions              Screen. . : 1
 4  Foreground    Interactive language processing        Language. : ENGLISH
 5  Batch         Submit job for language processing     Appl ID . : ISR
 6  Command       Enter TSO or Workstation commands      TSO logon : UPROC01
 7  Dialog Test   Perform dialog testing                 TSO prefix: TUTOR01
 8  LM Facility   Library administrator functions        System ID : UTEC203
 9  IBM Products  IBM program development products       MVS acct. : D1001
 10 SCLM          SW Configuration Library Manager       Release . : ISPF 6.3
 11 Workplace     ISPF Object/Action Workplace
 12 z/OS system   z/OS system programmer applications
 13 z/OS user     z/OS user applications
 D  DB2/DXT/QMF   Display DB2/DXT/QMF Selection Panel
 G  SWG/SVL       Display SWG/SVL Selection Panel

  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
```

   Depending on the customizations at your site, one of the following panels is displayed:

   - The **DB2I Primary Option Menu**
   - One or more customized selection panels

2. If any customized selection panels are displayed, select the options for Db2, and press Enter. For example, in the following panel, and Db2 is option 1.

```
------------------------ SITE SELECTION PANEL  -----------------------------
OPTION  ===> 1
                                                                    TUTOR01
   1  DB2         - Perform DATABASE 2 Interactive Functions
   2  DVR110 ADMIN- Invoke DataRefresher V1 Administrative Dialogs
   3  DVR110 EUSER- Invoke DataRefresher V1 End User Dialogs
   4  QMF         - QMF Query Management Facility
   5  QMFTOOL     - QMF Message Tool
   6  DB2PM       - DB2 Performance Monitor

Press END to return to ISPF/PDF Primary Option Menu.
```

The **DB2I Defaults** is displayed. Db2 Interactive (also known as "DB2I") is an interactive program that runs under TSO and uses ISPF services. You can use DB2I to interact with Db2 by entering values in panels.

3. Select the Db2 subsystem.
   a) From the **DB2I Primary Option Menu**, type option D in the **COMMAND** field and press Enter to display the **DB2I Defaults**.

```
                          DB2I PRIMARY OPTION MENU         SSID: DSN
COMMAND ===> D

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                  (Process SQL statements)
 2  DCLGEN                 (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION    (Prepare a DB2 application program to run)
 4  PRECOMPILE             (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE       (BIND, REBIND, or FREE plans or packages)
 6  RUN                    (RUN an SQL program)
 7  DB2 COMMANDS           (Issue DB2 commands)
 8  UTILITIES              (Invoke DB2 utilities)
 D  DB2I DEFAULTS          (Set global parameters)
 X  EXIT                   (Leave DB2I)

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

   b) In the **DB2I Defaults** panel, type the name of the Db2 subsystem that contains the sample tables in the **DB2 NAME** field, and press Enter.
   In the following example, the DB2A subsystem name is specified.

```
                          DB2I DEFAULTS PANEL 1
COMMAND ===>

Change defaults as desired:

 1  DB2 NAME .............. ===> DB2A      (Subsystem identifier)
 2  DB2 CONNECTION RETRIES ===> 0         (How many retries for DB2 connection)
 3  APPLICATION LANGUAGE   ===> IBMCOB    (ASM, C, CPP, IBMCOB, FORTRAN, PLI)
 4  LINES/PAGE OF LISTING  ===> 60        (A number from 5 to 999)
 5  MESSAGE LEVEL ........ ===> I         (Information, Warning, Error, Severe)
 6  SQL STRING DELIMITER   ===> DEFAULT   (DEFAULT, ' or ")
 7  DECIMAL POINT ........ ===> .         (. or ,)
 8  STOP IF RETURN CODE >= ===> 8         (Lowest terminating return code)
 9  NUMBER OF ROWS ....... ===> 20        (For ISPF Tables)
10  AS USER                ===>           (Userid to associate with the trusted
                                           connection)

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

   c) Press Enter again until you return to the **DB2I Primary Option Menu**.

   The **SSID** field contains the subsystem name that you specified. In the following example, shows that the DB2A subsystem is identified at the top of the panel.

```
                         DB2I PRIMARY OPTION MENU         SSID: DB2A
COMMAND ===>
...
```

4. From the **DB2I Primary Option Menu**, type option 1 in the **COMMAND** field and press Enter to open **SPUFI**.
5. In **SPUFI**, enter the following values:

   1 DATA SET NAME
   Specify the name of the data set and member to store the SQL statement that you will create in the next step. If the member does not already exist, Db2 creates it for you. In the following example, a partitioned data set named USER.SRCLIB.TUTORIAL is used, and the member name SQLTEXT is specified.

4 DATA SET NAME
Specify a name for your output data set. Db2 creates the output data set for you when you run the SQL statement. The output data set is qualified by your authorization ID.

5 CHANGE DEFAULTS
Type NO to prevent the **SPUFI Defaults** panel from being displayed. You can change this option later if you want to change the default values.

Fields 6 - 9
Ensure that these fields contain YES.

```
                          SPUFI                              SSID: DB2A
===>

Enter the input data set name:        (Can be sequential or partitioned)
 1  DATA SET NAME ... ===> 'USER.SRCLIB.TUTORIAL(SQLTEXT)'
 2  VOLUME SERIAL ... ===>            (Enter if not cataloged)
 3  DATA SET PASSWORD ===>            (Enter if password protected)

Enter the output data set name:       (Must be a sequential data set)
 4  DATA SET NAME ... ===> MYOUTPUT

Specify processing options:
 5  CHANGE DEFAULTS   ===> NO         (Y/N - Display SPUFI defaults panel?)
 6  EDIT INPUT ...... ===> YES        (Y/N - Enter SQL statements?)
 7  EXECUTE ......... ===> YES        (Y/N - Execute SQL statements?)
 8  AUTOCOMMIT ...... ===> YES        (Y/N - Commit after successful run?)
 9  BROWSE OUTPUT ... ===> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>
```

You are now ready to edit the data set or member.

6. Press Enter to open the **Edit** panel; then, type a SELECT statement.

   Tip: The ISPF editor works somewhat differently than text editors on platforms other than z/OS. If you are not familiar with this editor, see the following information:

   - The ISPF editor (z/OS Basic Skills)
   - Editing a data set (TSO/E Primer)

   The following statement selects every row of the sample EMP table, which contains sample employee data. For more information about how to create a SELECT statement, see subselect (Db2 SQL). For more information about the format of the sample EMP table and that data that it contains, see “Employee table (DSN8B10.EMP)” on page 322.

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       USER.SRCLIB.TUTORIAL(SQLTEXT) - 01.00           Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
''''''
''''''    SELECT * FROM DSN8B10.EMP;
''''''
''''''
''''''
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
  Menu  Help
```

7. When you finish typing the SELECT statement, press PF3 to redisplay the **SPUFI** panel.

   The SPUFI panel contains the following message:

```
                          SPUFI                              SSID: DB2A
===>
DSNE808A EDIT SESSION HAS COMPLETED. PRESS ENTER TO CONTINUE
```

8. Press Enter to issue the SQL statement.

   The **Browse** panel is displayed and shows the result of the SELECT statement.

```
 BROWSE    TUTOR01.MYOUTPUT                            Line 00000000 Col 001 080
 Command ===>                                            Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
 SELECT * FROM DSN8B10.EMP                                          00010011
                                                                    00020011
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   FIRSTNME      MIDINIT  LASTNAME         WORKDEPT  PHONENO  HIREDATE    J
---------+---------+---------+---------+---------+---------+---------+---------+
000010  CHRISTINE     I        HAAS             A00       3978     1965-01-01  P
000020  MICHAEL       L        THOMPSON         B01       3476     1973-10-10  M
000030  SALLY         A        KWAN             C01       4738     1975-04-05  M
000050  JOHN          B        GEYER            E01       6789     1949-08-17  M
000060  IRVING        F        STERN            D11       6423     1973-09-14  M
000070  EVA           D        PULASKI          D21       7831     1980-09-30  M
000090  EILEEN        W        HENDERSON        E11       5498     1970-08-15  M
000100  THEODORE      Q        SPENSER          E21       0972     1980-06-19  M
000110  VINCENZO      G        LUCCHESI         A00       3490     1958-05-16  S
000120  SEAN                   O'CONNELL        A00       2167     1963-12-05  C
000130  DOLORES       M        QUINTANA         C01       4578     1971-07-28  A
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Notice that the entire result set is not displayed in the panel. You can use the PF*n* keys that are listed at the bottom of the panel to scroll to see the rest of the data. For example, you can use PF8 to scroll down and PF11 to scroll right.

Tip: If SQL statements contain syntax errors or refer to object or column names that do not exist, the result might contain error messages or SQL codes. You can use the information in these messages and codes to troubleshoot the SQL statements. For more information about how to look up the meanings of messages and SQL codes, see “Introduction to troubleshooting problems in Db2 for z/OS” on page 201.

9. Press PF3 to redisplay the **SPUFI** panel, and press Enter.

   Tip: You can repeat steps 6 - 8 to issue successive SQL statements interactively from SPUFI.

   a. When you first press Enter from the **SPUFI** panel, the **Edit** panel is displayed. Add or revise your SQL statements.

   b. When you press PF3 to exit from the **Edit** panel, the **SPUFI** panel is redisplayed. The **6 EDIT INPUT** field now contains an asterisk (*), which indicates that the statements in the data set are ready to be issued.

   c. Press Enter again to run the statements. The results are shown in a **BROWSE** panel.

   d. Press PF3 to exit the **BROWSE** panel. The **SPUFI** panel is redisplayed, and field **6 EDIT INPUT** now contains YES. When you press Enter again, the **Edit** panel is displayed. You are now ready to edit the SQL statements again, as in step “10.a” on page 346.

   If you know that the data set already contains SQL statements that you want to run, you can skip the edit phase by changing the value of the **6 EDIT INPUT** field to NO. When you press Enter, the statement runs immediately and the results are shown in a **BROWSE** panel. For more information about working with SPUFI, see Executing SQL by using SPUFI (Db2 Application programming and SQL).

10. Comment out the original SELECT statement and type a new statement.

    a) Insert two hyphens (- -) to comment at the beginning of the statement to comment it out.

    b) Type the new statement several lines below the original. The new statement in the following example returns the employee numbers and calculates the total salary of each employee.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       USER.SRCLIB.TUTORIAL(SQLTEXT) - 01.12           Columns 00001 00072
 Command ===>                                                  Scroll ===> PAGE
 ****** ***************************** Top of Data ******************************
 000100   --SELECT * FROM DSN8B10.EMP;
 000200
 ''''''     SELECT EMPNO, SALARY + COMM AS TOTAL_SAL
 ''''''       FROM DSN8B10.EMP
 ''''''       ORDER BY TOTAL_SAL;
 ''''''
 ''''''
 ****** **************************** Bottom of Data ****************************
```

11. When you finish typing the new statement, press PF3 to return to **SPUFI**, and press Enter to issue the statement.

    The results are shown in a **BROWSE** panel.

```
   Menu  Utilities  Compilers  Help

 BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
 Command ===>                                                 Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
  --SELECT * FROM DSN8B10.EMP;                                          00010012
                                                                        00020011
    SELECT EMPNO, SALARY + COMM AS TOTAL_SAL                            00030012
      FROM DSN8B10.EMP                                                  00040013
      ORDER BY TOTAL_SAL;                                                00050012
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO      TOTAL_SAL
---------+---------+---------+---------+---------+---------+---------+---------+
000290      16567.00
000310      17172.00
200310      17172.00
000260      18630.00
000300      19170.00
000210      19732.00
000250      20714.00
000320      21546.00
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

    As before, the entire result set is not displayed on the panel. You can use the PF keys that are listed at the bottom of the screen to scroll to see the rest of the data.

What to do next

Press PF3 to return to the **Edit** panel.

Lesson checkpoint

In this lesson, you learned how to complete the following tasks interactively in SPUFI:

- Open SPUFI in ISPF
- Set default values in DB2I
- Issue SQL statements interactively and see the results

Related concepts

Common ways to interact with Db2 for z/OS

The z/OS operating system is ideal for processing *batch jobs*, workloads that run in the background with little or no human interaction. However, z/OS is as much an interactive operating system as it is a batch processing system.

Ways to select data from columns

Several techniques are available for selecting columns from a database for your result tables.

Related tasks

Submitting work by using DB2I (Db2 Administration Guide)

Lesson 1.2: Creating a table

You can create tables and other database objects interactively by issuing SQL statements from SPUFI. In this lesson, you will create a new table that has the same format and contains the same data as the sample EMP table that you queried in the previous lesson. In the next lesson, you can modify data in the new table that you created, without making any changes to the original sample data.

Before you begin

Before you begin

Complete “Lesson 1.1: Querying data interactively” on page 342. Lesson 1.2 builds upon what you learned in the previous lesson.

Procedure

Procedure

1. Open DB2I, set the subsystem ID, and open the USER.SRCLIB.TUTORIAL(SQLTEXT) data set in SPUFI as described in steps 1 - 5 of “Lesson 1.1: Querying data interactively” on page 342.
2. If your data set contains statements from the previous lesson, comment them out by adding two hyphens in front of them.
3. Create a table and add rows to it by issuing SQL statements.
   a) In the **Edit** panel, type a CREATE TABLE statement and an INSERT statement. For more information about the syntax and options of these SQL statements, see CREATE TABLE (Db2 SQL) and INSERT (Db2 SQL).
   For example, the following CREATE statement creates a table named MY_EMP under your authorization ID, which has the same format and contains the same data as the sample DSN8B10.EMP table. The following INSERT statement adds rows to the MY_EMP table. For more information about the data in the EMP table in the sample database, see “Employee table (DSN8B10.EMP)” on page 322.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       USER.SRCLIB.TUTORIAL(SQLTEXT) - 01.25          Columns 00001 00072
Command ===>                                                 Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000100
000101 --SELECT * FROM DSN8B10.EMP;
000102 --SELECT EMPNO, SALARY + COMM AS TOTAL_SAL;
000103   --FROM DSN8B10.EMP
000104   --ORDER BY TOTAL_SAL;
000110    CREATE TABLE MY_EMP LIKE DSN8B10.EMP;
000200    INSERT INTO MY_EMP
000300       SELECT * FROM DSN8B10.EMP;
000400
000500
000600
****** **************************** Bottom of Data ****************************

 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

   b) To issue the statements, press PF3, and then press Enter.
   The table is created, the rows are inserted, and the following result is displayed.

```
   Menu  Utilities  Compilers  Help

 BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
 Command ===>                                              Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
                                                                        00010023
--SELECT * FROM DSN8C10.EMP                                             00010125
--SELECT EMPNO, SALARY + COMM AS TOTAL_SAL                              00010225
--FROM DSN8C10.EMP                                                      00010326
--ORDER BY TOTAL_SAL                                                    00010426
   CREATE TABLE MY_EMP LIKE DSN8B10.EMP;                                00011027
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
   INSERT INTO MY_EMP                                                   00020027
      SELECT * FROM DSN8B10.EMP;                                        00030027
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 42
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Remember: In lesson 1.1, you learned that you can use any messages and codes that are issued to troubleshoot problems with your statements.

4. Press PF3 to return to **SPUFI**.

   For a reminder of the pattern that you follow to issue successive SQL statements from SPUFI, see the tip in Step 9 of the previous lesson.

5. Issue a SELECT statement to verify that the table was created and contains the data that you inserted.

   For example, the following query returns all rows from the table that you created:

```
   Menu  Utilities  Compilers  Help

 BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
 Command ===>                                              Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
                                                                        00010023
--SELECT * FROM DSN8B10.EMP                                             00010125
--SELECT EMPNO, SALARY + COMM AS TOTAL_SAL                              00010225
--FROM DSN8B10.EMP                                                      00010326
--ORDER BY TOTAL_SAL                                                    00010426
-- CREATE TABLE MY_EMP LIKE DSN8B10.EMP;                                00011029
-- INSERT INTO MY_EMP                                                   00020029
--    SELECT * FROM DSN8B10.EMP;                                        00030029
                                                                        00040026
   SELECT * FROM MY_EMP;                                                00050028
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   FIRSTNME      MIDINIT  LASTNAME         WORKDEPT  PHONENO  HIREDATE    J
---------+---------+---------+---------+---------+---------+---------+---------+
000010  CHRISTINE     I        HAAS             A00       3978     1965-01-01  P
000020  MICHAEL       L        THOMPSON         B01       3476     1973-10-10  M
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

What to do next

What to do next

Press PF3 and then press Enter to return to the Edit panel.

You will modify data in the MY_EMP table in the next lesson.

Lesson checkpoint

In this lesson, you learned how to complete the following tasks interactively in SPUFI:

- Create a table
- Insert rows from one table into another
- Verify that the new table was created and contains data

Related concepts

Creation of tables

Designing tables that many applications use is a critical task. Table design can be difficult because you can represent the same information in many different ways. This information briefly describes how tables are created and altered, and how authorization is controlled.

Related tasks

Loading data by using the INSERT statement (Db2 Administration Guide)

Lesson 1.3: Modifying data interactively

Just as you can query data in Db2 tables and to create database objects by issuing SQL statements in SPUFI, you can also insert, update, and delete data in Db2 tables. In this lesson, you will add a row for a new employee to the MY_EMP table that you created in the previous lesson, modify a value in the row, and finally delete the new row. Along the way, you will learn how to verify the result of each of the actions, all by using SPUFI.

Before you begin

Before you begin

You created the MY_EMP table under your authorization ID, as described in "Lesson 1.2: Creating a table" on page 348.

About this task

Procedure

Procedure

1. Open DB2I, set the subsystem ID, and open the USER.SRCLIB.TUTORIAL(SQLTEXT) data set in SPUFI as described in steps 1 - 5 of "Lesson 1.1: Querying data interactively" on page 342.
2. Delete the SQL statements that you entered in the previous lessons.

   You can delete rows from the data set by issuing the D*n* line command, where *n* is the number of lines to delete.

   For example, the following command deletes nine rows from the data set:

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       USER.SRCLIB.TUTORIAL(SQLTEXT) - 01.29           Columns 00001 00072
Command ===>                                                 Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000100
00d901 --SELECT * FROM DSN8B10.EMP
000102 --SELECT EMPNO, SALARY + COMM AS TOTAL_SAL
000103 --FROM DSN8C10.EMP
000104 --ORDER BY TOTAL_SAL
000110 -- CREATE TABLE MY_EMP LIKE DSN8B10.EMP;
000200 -- INSERT INTO MY_EMP
000300 --    SELECT * FROM DSN8B10.EMP;
000400
000500 -- SELECT * FROM MY_EMP;
000600
****** **************************** Bottom of Data ****************************

 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

3. Suppose that your company is preparing to hire a new employee. You must add a row of data to the MY_EMP table for the new employee.

   In the following example INSERT statement, each item in the first set of parentheses is a column name in the MY_EMP table. Each value in the second set of parentheses is the value that will be added to the corresponding column. For example, 300010 will be added to the EMPNO column, VICTORIA will be added to the FIRSTNME column, and so on. For more information about the syntax and options of an INSERT statement, see INSERT (Db2 SQL).

Remember that you created the MY_EMP table in the previous lesson, based on the format of the sample EMP table. For detailed information about the format and contents of both tables, see “Employee table (DSN8B10.EMP)” on page 322.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       USER.SRCLIB.TUTORIAL(SQLTEXT) - 01.30          Columns 00001 00072
Command ===>                                                Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000100   INSERT INTO MY_EMP (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
000600                       WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
000610                       SEX, BIRTHDATE, SALARY, BONUS, COMM)
000620   VALUES ('300010', 'VICTORIA', 'M', 'OFALLON', 'D11', '5240',
000630          '2014-04-21', 'DESIGNER', 16, 'F', '1996-07-03', 30000.00,
000640          300.00, 0.00);
000700
****** **************************** Bottom of Data ****************************
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

4. Press PF3 and then press Enter to issue the statement.

   The results are displayed in a **BROWSE** panel:

```
  Menu  Utilities  Compilers  Help

BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
Command ===>                                                Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
  INSERT INTO MY_EMP (EMPNO, FIRSTNME, MIDINIT, LASTNAME,               00010030
                WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,              00060030
                SEX, BIRTHDATE, SALARY, BONUS, COMM)                    00061030
  VALUES ('300010', 'VICTORIA', 'M', 'OFALLON', 'D11', '5240',          00062030
       '2014-04-21', 'DESIGNER', 16, 'F', '1996-07-03', 30000.00,       00063030
       300.00, 0.00)                                                    00064030
                                                                        00070023
                                                                        00080023
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

   Remember: In lesson 1.1, you learned that you can use any messages and codes that are issued to troubleshoot problems with your statements.

5. Verify that the row was inserted by issuing a SELECT statement to query the MY_EMP table.
   The SELECT statement in the following example includes a WHERE clause that returns only the new row that you added.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       USER.SRCLIB.TUTORIAL(SQLTEXT) - 01.30          Columns 00001 00072
Command ===>                                                Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000100 --INSERT INTO MY_EMP (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
000600 --              WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
000610 --              SEX, BIRTHDATE, SALARY, BONUS, COMM)
000620 --VALUES ('300010', 'VICTORIA', 'M', 'OFALLON', 'D11', '5240',
000630 --     '2014-04-21', 'DESIGNER', 16, 'F', '1996-07-03', 30000.00,
000640 --     300.00, 0.00);
000700
000800   SELECT * FROM MY_EMP WHERE EMPNO='300010';
****** **************************** Bottom of Data
****************************
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

   The following result is returned:

```
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   FIRSTNME      MIDINIT  LASTNAME         WORKDEPT  PHONENO  HIREDATE    J
---------+---------+---------+---------+---------+---------+---------+---------+
300010  VICTORIA      M        OFALLON          D11       5240     2014-04-21  D
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
```

6. Suppose that the new employee has excellent negotiation skills, so that the company has decided to increase her salary offer by ten percent. Issue an UPDATE statement to increase the salary of the employee that you added.
   For more information about the syntax and options of an UPDATE statement, see UPDATE (Db2 SQL).

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       USER.SRCLIB.TUTORIAL(SQLTEXT) - 01.32           Columns 00001 00072
 Command ===>                                                  Scroll ===> PAGE
 ****** ***************************** Top of Data ******************************
 000100 --INSERT INTO MY_EMP (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
 000600 --              WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,
 000610 --              SEX, BIRTHDATE, SALARY, BONUS, COMM)
 000620 --VALUES ('300010', 'VICTORIA', 'M', 'OFALLON', 'D11', '5240',
 000630 --     '2014-04-21', 'DESIGNER', 16, 'F', '1996-07-03', 30000.00,
 000640 --     300.00, 0.00);
 000700 -- SELECT * FROM MY_EMP WHERE EMPNO='300010';
 ''''''
 000710    UPDATE MY_EMP
 000720      SET SALARY = SALARY + 3000
 000730      WHERE EMPNO = '300010';
 000800
 ****** **************************** Bottom of Data ****************************
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

   The following messages indicate that the UPDATE statement was successful:

```
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
```

7. Verify that the change was applied by issuing a SELECT statement.
   For example, you might issue a statement that returns the following result:

```
   Menu  Utilities  Compilers  Help

 BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
 Command ===>                                           Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
--INSERT INTO MY_EMP (EMPNO, FIRSTNME, MIDINIT, LASTNAME,              00010031
--              WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL,            00060031
--              SEX, BIRTHDATE, SALARY, BONUS, COMM)                  00061031
--VALUES ('300010', 'VICTORIA', 'M', 'OFALLON', 'D11', '5240',        00062031
--     '2014-04-21', 'DESIGNER', 16, 'F', '1996-07-03', 30000.00,     00063031
--     300.00, 0.00);                                                 00064031
   SELECT EMPNO, LASTNAME, FIRSTNME, SALARY FROM MY_EMP               00070034
                WHERE EMPNO='300010';                                 00070134
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   LASTNAME         FIRSTNME          SALARY
---------+---------+---------+---------+---------+---------+---------+---------+
300010  OFALLON          VICTORIA         31000.00
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

8. Despite the increased salary offer, the new employee decided to take a job offer elsewhere. Issue a DELETE statement to remove the row that you added for the new employee.
   The following example shows the DELETE statement to remove the row and the result of issuing the DELETE statement. For more information about the syntax and options of a DELETE statement, see DELETE (Db2 SQL).

```
   Menu  Utilities  Compilers  Help

 BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
 Command ===>                                              Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
 DELETE FROM MY_EMP                                                    00010012
   WHERE EMPNO = '300010';                                             00011012
                                                                       00020012
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 3
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 15
****************************** Bottom of Data ******************************

  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

9. Verify that the row was deleted by issuing a SELECT statement.
   For example, you might issue a SELECT statement that returns that following result:

```
   Menu  Utilities  Compilers  Help

 BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
 Command ===>                                              Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
 SELECT * FROM MY_EMP                                                  00010013
   WHERE EMPNO = '300010';                                             00011012
                                                                       00020012
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   FIRSTNME      MIDINIT  LASTNAME         WORKDEPT  PHONENO  HIREDATE    J
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE610I NUMBER OF ROWS DISPLAYED IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 3
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 17
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

   The DSNE610I message indicates that 0 rows were returned, which means that employee number 300010 is no longer included in the MY_EMP table.

Lesson checkpoint
In this lesson, you learned how to complete the following tasks interactively in SPUFI:

- Insert a row of data
- Update a row of data
- Delete a row of data
- Verify the results of the actions

Module 2: Running Db2 utilities

IBM Db2 utilities are tools that help maintain data in your Db2 for z/OS databases. Generally, you use Db2 utilities to perform large-scale operations across one or more table spaces. You can use several different methods to run Db2 utilities. This module shows you two of these methods: using a Db2 Interaction (DB2I) panel and writing job control language (JCL) jobs.

Learning objectives

In this module, you will learn how to do the following tasks:

- Run utilities by using a DB2I panel
- Run utilities by writing and submitting JCL jobs
- View the results of utility jobs

Time required
1 hour

Before you begin

- Make sure that you complete the prerequisites that are listed in the introduction to this tutorial.
- Verify that the Db2 Utilities Suite for z/OS (5655-W87, FMID JDBBB1K) is installed on your system.
- Obtain authority to run the COPY, LOAD, and UNLOAD utilities. For a list of required authorizations for these utilities, see the following information:
  - COPY (Db2 Utilities)
  - LOAD (Db2 Utilities)
  - UNLOAD (Db2 Utilities)
- Create or obtain access to a partitioned data set to contain your work. Contact your system administrator to find out the conventions for data sets at your site, or accept the default values. This tutorial assumes that the data set is named USER.SRCLIB.TUTORIAL.
- Log on to z/OS by using a TSO session.
- Create the MY_EMP table under your authorization ID and populate it, as described in "Lesson 1.2: Creating a table" on page 348.

Lesson 2.1: Running the UNLOAD utility from DB2I

Db2 Interactive (DB2I) is a quick way to run a utility without knowing job control language (JCL). You can run most online utilities from the **DB2 Utilities** panel in DB2I.

For this lesson, suppose that you want to copy your data from the MY_EMP table into another table so that you can use that copy for testing purposes. In this lesson, you will use the UNLOAD utility to copy the data in the MY_EMP table into an output data set. Then, in the next lesson, you will load that data into another table.

The UNLOAD utility copies the data in a format that can be used by the LOAD utility. UNLOAD does not delete the data from the source object, so the original table remains unchanged. You can also request that UNLOAD generate a LOAD statement based on the data, which can save you the time of having to create the statement yourself.

Before you begin
Before you begin

Complete the prerequisites that are described in "Module 2: Running Db2 utilities" on page 353.

Procedure

Procedure

1. Open **DB2I Primary Option Menu**, and set the subsystem ID as described in steps 1 - 3 of "Lesson 1.1: Querying data interactively" on page 342.
2. Determine the database name and table space name of the MY_EMP table.

For example, if you have SELECT privilege for the Db2 catalog tables, you might issue the following query to return the database name and table space name of your table:

```
SELECT NAME, CREATOR, DBNAME, TSNAME
  FROM SYSIBM.SYSTABLES
    WHERE NAME='MY_EMP';
```

The result contains the database name in the DBNAME column and the table space name in the TSNAME column. If more than one MY_EMP table exists, use the CREATOR column to identify the row for the table that you created.

If you do not have SELECT privilege for the Db2 catalog, ask a colleague who does have SELECT privilege for the Db2 catalog tables to get this information for you by querying the catalog.

3. Create another member in the USER.SRCLIB.TUTORIAL data set.
   a) From the **ISPF/PDF Primary Option Menu**, select option 2 EDIT.
   b) In the **Edit Entry** panel, type the data set name and the new member name in the **Name** field, and press Enter.

```
  Menu  RefList  RefMode  Utilities  Workstation  Help

                              Edit Entry Panel
Command ===>
                                                                More:     +
ISPF Library:
   Project . . .
   Group . . . .          . . .          . . .          . . .
   Type  . . . .
   Member  . . .                (Blank or pattern for member selection list)

Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file:
   Name . . . . . 'USER.SRCLIB.TUTORIAL(MYUTIL)'                          +
   Volume Serial . .           (If not cataloged)

Workstation File:
   File Name  . .
                                     Options
Initial Macro  . . . .                  Confirm Cancel/Move/Replace
Profile Name . . . . .                  Mixed Mode
Format Name  . . . . .                  Edit on Workstation
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

   The **Edit** panel is displayed for the new member.

4. Type an UNLOAD utility control statement to unload the MY_EMP table. Then, press PF3 to save and exit.

   You can use the following example UNLOAD utility statement. Change the qualified table space name (DSN00006.MYREMP) to the values that you noted in step 2 and the table qualifier (ADMF002) to your user ID.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       USER.SRCLIB.TUTORIAL(MYUTIL) - 01.13            Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000100  UNLOAD TABLESPACE DSN00006.MYREMP
000200  FROM TABLE ADMF002.MY_EMP
****** **************************** Bottom of Data ****************************
```

   Tip: If you need help with the UNLOAD utility syntax, see Syntax and options of the UNLOAD control statement (Db2 Utilities)

5. Navigate to the **DB2I Primary Option Menu**.

   If you need help navigating to this panel, see “Lesson 1.1: Querying data interactively” on page 342.

6. From the **DB2I Primary Option Menu**, select option 8 to display the **DB2 Utilities** panel.
7. Specify some basic information about the utility that you want to run, and press Enter:

3 UTILITY
The name of the utility that you want to run. In this case, you want to run UNLOAD.

4 STATEMENT DATA SET
The name of the data set or member in which your utility control statement is stored. If you are using the same data set names as the tutorial, this value is `'USER.SRCLIB.TUTORIAL(MYUTIL)'`. In this case, the value is enclosed in single quotation marks so that your user name is not appended to the front.

6 LISTDEF and TEMPLATE
Use these fields to indicate whether you want to use lists (defined by the LISTDEF utility) and templates (defined by the TEMPLATE utility). These fields are blank by default, and you must specify a value for them. In this case, you do not want to use lists or templates, so type NO in these fields.

Tip: You can use the Tab key to navigate between fields.

```
                          DB2 UTILITIES                    SSID: DB2A
===>

Select from the following:

 1 FUNCTION ===> EDITJCL            (SUBMIT job, EDITJCL, DISPLAY, TERMINATE)
 2 JOB ID   ===> TEMP               (A unique job identifier string)
 3 UTILITY  ===> UNLOAD             (CHECK DATA, CHECK INDEX, CHECK LOB,
                                     COPY, DIAGNOSE, LOAD, MERGE, MODIFY,
                                     QUIESCE, REBUILD, RECOVER, REORG INDEX,
                                     REORG LOB, REORG TABLESPACE, REPORT,
                                     REPAIR, RUNSTATS, STOSPACE, UNLOAD)
 4 STATEMENT DATA SET ===> 'USER.SRCLIB.TUTORIAL(MYUTIL)'

 5 RESTART  ===> NO                 (NO, CURRENT, PHASE or PREVIEW)

 6 LISTDEF? (YES|NO) ===> NO        TEMPLATE? (YES|NO) ===> NO

 7 LIB ==>                                 (BLANK or DB2 Library name).

 * The data set names panel will be displayed when required by a utility.

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

8. In the **Data Set Names** panel, specify the data sets to be used by the utility, and press Enter.

1 RECDSN
The name of the data set where you want to store the data that is unloaded from the table. Db2 allocates this data set for you.

7 PUNCHDSN
The name of the data set where you want to store the generated LOAD statement. Db2 allocates this data set for you.

```
                DATA SET NAMES                      SSID:
===>

Enter data set name for LOAD, REORG TABLESPACE, or UNLOAD:
 1 RECDSN   ===> UNLOAD.MYEMP

Enter data set name for LOAD or REORG TABLESPACE:
 2 DISCDSN  ===>

Enter output data sets for local/current site for COPY, MERGECOPY,
  LOAD, or REORG TABLESPACE:
 3 COPYDSN  ===>
 4 COPYDSN2 ===>

Enter output data sets for recovery site for COPY, LOAD, or REORG
  TABLESPACE:
 5 RCPYDSN1 ===>
 6 RCPYDSN2 ===>

Enter output data sets for REORG or UNLOAD
 7 PUNCHDSN ===> SYSPUNCH.MYEMP

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

The following output is displayed to confirm your request to run the utility:

```
 >>DSNU EXEC:
 >>  UNLOAD UTILITY REQUESTED WITH
 >>    CONTROL=NONE, EDIT=SPF, RESTART=NO,
 >>    INDSN=USER.SRCLIB.TUTORIAL(MYUTIL), RECDSN=TUTOR01.UNLOAD.MYEMP, PUNCHDSN
=TUTOR01.SYSPUNCH.MYEMP,
 >>    DISCDSN="OMITTED",LISTDSN="OMITTED", TEMPDSN="OMITTED",
 >>    COPYDSN=**NOT REQUIRED**, COPYDSN2=**NOT REQUIRED**,
 >>    RCPYDSN1=**NOT REQUIRED**, RCPYDSN2=**NOT REQUIRED**,
 >>    SYSTEM=DB2A, SUBMIT=NO, UID=TEMP,
 >>    UNIT=SYSDA, VOLUME="OMITTED", DB2I=YES,
 >>    LIB="NULL".                                      /*@041*/
 >>  THE RESULTING JCL WILL BE WRITTEN TO DSNUUNL.CNTL
 >>SPF EDITING FACILITY INVOKED TO EDIT DSNUUNL.CNTL
 >>  WHEN *** APPEAR, PLEASE PRESS ENTER
 >>  TO TERMINATE SPF:
 >>     PRESS PF3    - RETURN TO CLIST WITH CHANGES
 >>     PRESS PF4    - RETURN TO CLIST WITH CHANGES THEN
 >>                    RETURN TO MAIN MENU
 >>     ENTER CANCEL - RETURN TO CLIST WITH NO CHANGES
 ***
```

If any error messages are issued, press Enter to return to the **DB2 Utilities** panel and correct the mistakes.

Otherwise, press Enter again to generate the utility job. The job is displayed in an **EDIT** panel.

9. In the **EDIT** panel, press PF8 to scroll down.

   The generated JCL job contains the UNLOAD utility control statement that you created.

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       TUTOR01.DSNUUNL.CNTL                          Columns 00001 00072
 Command ===>                                                Scroll ===> PAGE
 000016 //     SPACE=(16384,(20,20),,,ROUND),
 000017 //     UNIT=SYSDA
 000018 //DSNUPROC.SYSPUNCH DD DSN=TUTOR01.SYSPUNCH.MYEMP,
 000019 //     DISP=(MOD,CATLG),
 000020 //     SPACE=(16384,(20,20),,,ROUND),
 000021 //     UNIT=SYSDA
 000022 //DSNUPROC.SYSIN    DD  *
 000023  UNLOAD TABLESPACE DSN00006.MYREMP
 000024  FROM TABLE TUTOR01.MY_EMP
 000025 //
 ****** **************************** Bottom of Data ****************************
```

10. To run the JCL utility job, type SUB in the **Command** field and press Enter.

SUB is the TSO SUBMIT command, which submits the data set or member to be processed as a batch job.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       TUTOR01.DSNUUNL.CNTL                            Columns 00001 00072
Command ===>  SUB                                             Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000001 //TUTOR01A JOB (ACCOUNT),'NAME'
000002 //*
000003 //*
000004 //*
000005 //UTIL EXEC DSNUPROC,SYSTEM=DB2A,UID='TEMP',UTPROC=''
000006 //*
000007 //*********************************************
000008 //*
000009 //*  GENERATING JCL FOR THE UNLOAD UTILITY
000010 //*  DATE:  04/28/14          TIME:  14:43:14
000011 //*
000012 //*********************************************
000013 //*
000014 //DSNUPROC.SYSREC DD DSN=TUTOR01.UNLOAD.MYEMP,
000015 //     DISP=(MOD,CATLG),
000016 //     SPACE=(16384,(20,20),,,ROUND),
000017 //     UNIT=SYSDA
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

If you receive the message `ENTER JOBNAME CHARACTER(S) -`, type any character and press Enter.

A message indicates that the job was submitted and provides the job number. Note the job number for future use, and press Enter.

```
JOB TUTOR01A(JOB00064) SUBMITTED
***
```

11. To review the output, type `TSO SDSF H` in the **Command** field.

```
EDIT       TUTOR01.DSNUUNL.CNTL                          Columns 00001 00072
Command ===>  TSO SDSF H                                    Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000001 //TUTOR01A JOB (ACCOUNT),'NAME'
000002 //*
```

The **SDSF HELD OUTPUT DISPLAY** opens, which contains a list of held jobs in z/OS System Display and Search Facility (SDSF).

Tip: If you need help navigating between SDSF panels, see What is SDSF? (z/OS basic skills).

12. Display the results of the job. In the **NP** column, type S in the row for your job number, and press Enter.

```
SDSF HELD OUTPUT DISPLAY ALL CLASSES LINES 2,049       LINE 1-5 (5)
COMMAND INPUT ===>                                     SCROLL ===> PAGE
NP   JOBNAME  JobID    Owner    Prty C ODisp Dest              Tot-Rec  Tot-
     TUTOR01  TSU00057 ADMF002   144 Z HOLD  LOCAL                 498
     TUTOR01  TSU00059 ADMF002   144 Z HOLD  LOCAL                 419
     TUTOR01  TSU00060 ADMF002   144 Z HOLD  LOCAL                 431
     TUTOR01  TSU00061 ADMF002   144 Z HOLD  LOCAL                 425
S    TUTOR01A JOB00064 ADMF002   144 Z HOLD  LOCAL                 276
```

The utility output is displayed. In the output, the utility return code `DSNUTILB 0000` indicates that the utility completed with no warnings or errors.

```
 SDSF OUTPUT DISPLAY ADMF002A JOB00064  DSID     2 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===>                                           SCROLL ===> PAGE
******************************** TOP OF DATA *********************************
                    J E S 2  J O B  L O G  --  S Y S T E M  S T L 0  --  N O D

14.55.06 JOB00064 ---- MONDAY,    28 APR 2014 ----
14.55.06 JOB00064  IRR010I  USERID ADMF002  IS ASSIGNED TO THIS JOB.
14.55.06 JOB00064  ICH70001I TUTOR01  LAST ACCESS AT 14:13:49 ON MONDAY, APRIL 2
14.55.06 JOB00064  $HASP373 TUTOR01A STARTED - INIT 8    - CLASS A - SYS STL0
14.55.08 JOB00064  SMF000I  TUTOR01A    DSNUPROC    DSNUTILB    0000
14.55.08 JOB00064  $HASP395 TUTOR01A ENDED
------ JES2 JOB STATISTICS ------
  28 APR 2014 JOB EXECUTION DATE
           24 CARDS READ
          276 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
           18 SYSOUT SPOOL KBYTES
         0.03 MINUTES EXECUTION TIME
        1 //TUTOR01A JOB (ACCOUNT),'NAME'
          //*
          //*
          //*
        2 //UTIL EXEC DSNUPROC,SYSTEM=DB2A,UID='TEMP',UTPROC=''
        3 XXDSNUPROC PROC LIB='DB2A.DSNLOAD',
```

You can type BOTTOM in the **COMMAND INPUT** line to jump to the bottom of the data and see the utility messages. You can also use PF11 to scroll right and PF10 to scroll left.

```
 SDSF OUTPUT DISPLAY ADMF002A JOB00064  DSID   102 LINE 1       COLUMNS 02- 81
 COMMAND INPUT ===>                                           SCROLL ===> PAGE
DSNU000I    118 14:55:07.75 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = TEMP
DSNU1044I   118 14:55:07.90 DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    118 14:55:08.03 DSNUGUTC -  UNLOAD TABLESPACE DSN00006.MYREMP
DSNU650I  -DB2A 118 14:55:08.20 DSNUUGMS -   FROM TABLE TUTOR01.MY_EMP
DSNU253I    118 14:55:08.55 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECOR
DSNU252I    118 14:55:08.55 DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECOR
DSN00006.MYREMP
DSNU250I    118 14:55:08.55 DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00
DSNU010I    118 14:55:08.62 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETUR
******************************** BOTTOM OF DATA ********************************
```

13. Press PF3 until you return to the **z/OS Primary Option Menu**.
14. Confirm that unload data set was created.
    a) From **ISPF/PDF Primary Option Menu**, type 3 . 4 and press Enter to display the **Data Set List Utility** panel.
    b) In the **Dsname Level** field, type your authorization ID, ensure that the **Include Additional Qualifiers** field is selected, and press Enter.

```
  Menu  RefList  RefMode  Utilities  Help

                           Data Set List Utility
Option ===>
                                                                    More:     +
   blank Display data set list              P Print data set list
       V Display VTOC information          PV Print VTOC information

Enter one or both of the parameters below:
   Dsname Level . . . TUTOR01
   Volume serial  . .

Data set list options
   Initial View               Enter "/" to select option
   1  1. Volume               /  Confirm Data Set Delete
      2. Space                /  Confirm Member Delete
      3. Attrib               /  Include Additional Qualifiers
      4. Total                /  Display Catalog Name
                                 Display Total Tracks
                                 Prefix Dsname Level

When the data set list is displayed, enter either:
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

A list of your data sets is displayed. The list includes the data set that your UNLOAD utility job created to contain the unloaded data. In this example, the unload data set is TUTOR01.UNLOAD.MYEMP.

```
DSLIST - Data Sets Matching ADMF002                              Row 1 of 5
Command ===>                                                  Scroll ===> PAGE

Command - Enter "/" to select action                Message          Volume
------------------------------------------------------------------------------
         TUTOR01.DSNUUNL.CNTL                                        SCR03
         TUTOR01.MYOUT                                               SCR03
         TUTOR01.SPFLOG1.LIST                                        SCR03
         TUTOR01.SYSPUNCH.MYEMP                                      SCR03
         TUTOR01.UNLOAD.MYEMP                                        SCR03
***************************** End of Data Set list ****************************
```

You can look at this data set to confirm that it contains the expected data. If you do so, you might notice some unreadable characters. Such characters are expected. They represent items such as null-indicator bytes and numeric fields in internal formats.

15. Open the SYSPUNCH data set for editing. This data set contains the generated LOAD statement. In the **Command** column, type E in the row for the TUTOR01.SYSPUNCH.MYEMP data set, and press Enter twice.

```
DSLIST - Data Sets Matching ADMF002                              Row 1 of 5
Command ===>                                                  Scroll ===> PAGE

Command - Enter "/" to select action                Message          Volume
------------------------------------------------------------------------------
         TUTOR01.DSNUUNL.CNTL                                        SCR03
         TUTOR01.MYOUT                                               SCR03
         TUTOR01.SPFLOG1.LIST                                        SCR03
   E     TUTOR01.SYSPUNCH.MYEMP                                      SCR03
         TUTOR01.UNLOAD.MYEMP                                        SCR03
***************************** End of Data Set list ****************************
```

An **EDIT** panel displays the contents of the SYSPUNCH data set.

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       TUTOR01.SYSPUNCH.MYEMP                        Columns 00001 00072
 Command ===>                                                Scroll ===> PAGE
 ****** ***************************** Top of Data ******************************
 000001 LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 000002  EBCDIC  CCSID(00037,00000,00000)
 000003  INTO TABLE
 000004  "TUTOR01".
 000005  "MY_EMP"
 000006  WHEN(00001:00002) = X'0003'
 000007  NUMRECS                  43
 000008  ( "EMPNO"
 000009   POSITION(  00003:00008) CHAR(00006)
 000010  , "FIRSTNME"
 000011   POSITION(  00009:00022) VARCHAR
 000012  , "MIDINIT"
 000013   POSITION(  00023:00023) CHAR(00001)
 000014  , "LASTNAME"
 000015   POSITION(  00024:00040) VARCHAR
 000016  , "WORKDEPT"
 000017   POSITION(  00042:00044) CHAR(00003)
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

What to do next

Leave the **EDIT** panel open. In the next lesson, you will modify this LOAD statement.

Lesson checkpoint
In this lesson, you learned how to complete the following tasks:

- Run a utility from DB2I
- View the utility job results
- View the output data sets

Lesson 2.2: Running the LOAD utility from DB2I

You can use the LOAD utility to load data into one or more tables. You can also insert rows by using SQL INSERT statements, but LOAD is a more efficient way to add a large number of rows.

Recall that in the previous lesson ("Lesson 2.1: Running the UNLOAD utility from DB2I" on page 354), you copied data from the MY_EMP table by using the UNLOAD utility. In this lesson, you will load that data into another table that you can use for testing. Although you can run the LOAD utility by using other methods, you will continue to use Db2 Interactive (DB2I) in this lesson.

Before you begin
Before you begin

Complete "Lesson 2.1: Running the UNLOAD utility from DB2I" on page 354, and open the TUTOR01.SYSPUNCH.MYEMP data set.

Procedure
Procedure

1. Edit the TUTOR01.SYSPUNCH.MYEMP data set and change the table name in the LOAD utility control statement.
   In the following example, the name has been changed to MY_EMP2.

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       TUTOR01.SYSPUNCH.MYEMP                          Columns 00001 00072
 Command ===>                                                  Scroll ===> PAGE
 ****** ***************************** Top of Data ******************************
 000001 LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 000002  EBCDIC  CCSID(00037,00000,00000)
 000003  INTO TABLE
 000004  "TUTOR01".
 000005  "MY_EMP2"
 000006  WHEN(00001:00002) = X'0003'
 000007  NUMRECS                  43
 000008  ( "EMPNO"
 000009   POSITION(  00003:00008) CHAR(00006)
 000010  , "FIRSTNME"
 000011   POSITION(  00009:00022) VARCHAR
 000012  , "MIDINIT"
 000013   POSITION(  00023:00023) CHAR(00001)
 000014  , "LASTNAME"
 000015   POSITION(  00024:00040) VARCHAR
 000016  , "WORKDEPT"
 000017   POSITION(  00042:00044) CHAR(00003)
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

2. Press PF3 to exit. Then, navigate to **SPUFI**, and issue the following CREATE TABLE statement to create the table that you specified in the LOAD utility control statement:

```
CREATE TABLE MY_EMP2 LIKE MY_EMP;
```

   If you are not familiar with SPUFI or need a refresher on using SPUFI, see "Lesson 1.2: Creating a table" on page 348.

   A new empty table is created.
3. From the **DB2I Primary Option Menu**, select option 8 to open the **DB2 Utilities** panel.
4. Specify some basic information about the utility that you want to run, and press Enter:

3 UTIILITY
The name of the utility that you want to run. This time, you want to run LOAD.

4 STATEMENT DATA SET
The name of the data set or member that contains the utility control statement. In this case, you want to use the LOAD statement that the UNLOAD utility generated in the previous lesson. If you are following the conventions of this tutorial, that data set is named *userID*.SYSPUNCH.MYEMP, where *userID* is your Db2 authorization ID.

6 LISTDEF and TEMPLATE
As in the previous lesson, you do not want to use the LISTDEF or TEMPLATE utilities. Type NO in these required fields.

```
                            DB2 UTILITIES                    SSID: DB2A
===>

Select from the following:

 1 FUNCTION ===> EDITJCL            (SUBMIT job, EDITJCL, DISPLAY, TERMINATE)
 2 JOB ID   ===> TEMP               (A unique job identifier string)
 3 UTILITY  ===> LOAD               (CHECK DATA, CHECK INDEX, CHECK LOB,
                                     COPY, DIAGNOSE, LOAD, MERGE, MODIFY,
                                     QUIESCE, REBUILD, RECOVER, REORG INDEX,
                                     REORG LOB, REORG TABLESPACE, REPORT,
                                     REPAIR, RUNSTATS, STOSPACE, UNLOAD)
 4 STATEMENT DATA SET ===> 'TUTOR01.SYSPUNCH.MYEMP'

 5 RESTART  ===> NO                 (NO, CURRENT, PHASE or PREVIEW)

 6 LISTDEF? (YES|NO) ===> NO        TEMPLATE? (YES|NO) ===> NO

 7 LIB ==>                                 (BLANK or DB2 Library name).

 * The data set names panel will be displayed when required by a utility.

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

5. In the **DATA SET NAMES** panel, specify the data sets to be used by the utility, and generate the job:
   a) In field **1 RECDSN**, specify the name of the data set that contains the data that you want to load. In this case, you want to load the data that you unloaded from the MY_EMP table in the previous lesson. The name of that unload data set is UNLOAD.MYEMP.
   b) Ensure that field **7 PUNCHDSN** is blank.
   c) Press Enter twice to generate the utility job.

```
               DATA SET NAMES                       SSID:
===>

Enter data set name for LOAD, REORG TABLESPACE, or UNLOAD:
 1 RECDSN   ===> UNLOAD.MYEMP

Enter data set name for LOAD or REORG TABLESPACE:
 2 DISCDSN  ===>

Enter output data sets for local/current site for COPY, MERGECOPY,
  LOAD, or REORG TABLESPACE:
 3 COPYDSN  ===>
 4 COPYDSN2 ===>

Enter output data sets for recovery site for COPY, LOAD, or REORG
  TABLESPACE:
 5 RCPYDSN1 ===>
 6 RCPYDSN2 ===>

Enter output data sets for REORG or UNLOAD
 7 PUNCHDSN ===>

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

The generated job is displayed. Scroll down to see that it includes the LOAD statement from the TUTOR01.SYSPUNCH.MYEMP data set.

6. To run the JCL utility job, type SUB in the **Command** field and press Enter.

```
EDIT       TUTOR01.DSNUUNL.CNTL                            Columns 00001 00072
 Command ===> SUB                                             Scroll ===> PAGE
 ****** ***************************** Top of Data ******************************
 ==MSG> -Warning- The UNDO command is not available until you change
 ==MSG>           your edit profile using the command RECOVERY ON.
 000001 //ADMF002A JOB (ACCOUNT),'NAME'
 000002 //*
 000003 //*
 000004 //*
 000005 //UTIL EXEC DSNUPROC,SYSTEM=DB2A,UID='TEMP',UTPROC=''
 000006 //*
 000007 //*********************************************
 000008 //*
 000009 //*  GENERATING JCL FOR THE UNLOAD UTILITY
 000010 //*  DATE:  04/25/14          TIME:  10:47:06
 000011 //*
 000012 //*********************************************
 000013 //*
 000014 //DSNUPROC.SYSREC DD DSN=ADMF002.UNLOAD.MYEMP,
 000015 //     DISP=(MOD,CATLG)
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

If you receive the message `ENTER JOBNAME CHARACTER(S) -`, type any character and press Enter.

A message indicates that the job was submitted and provides the job number. Note the job number for future use, and press Enter.

```
JOB TUTOR01(JOB00161) SUBMITTED
 ***
```

7. Review the output. Type `TSO SDSF H` in the **Command** field and press Enter.

   The **SDSF HELD OUTPUT DISPLAY** opens, which contains a list of held jobs.

8. Display the results of the job. In the **NP** column, type S in the row for your job number, and press Enter.

   The utility output is shown, and the utility return code `DSNUTILB 0004` indicates that the utility completed with warning messages.

9. Determine the reason for the warning return code. Type `BOTTOM` in the **Command** field, and press Enter. Press PF11 to scroll right.

   In the following example, you can see part of message DNSU380I, which indicates the table space was left in COPY PENDING status.

```
 SDSF OUTPUT DISPLAY TUTOR01A JOB00161  DSID   102 LINE 6       COLUMNS 55- 134
 COMMAND INPUT ===>                                           SCROLL ===> PAGE
E" POSITION(9:22) VARCHAR,
" POSITION(23:23) CHAR(1),
E" POSITION(24:40) VARCHAR,
T" POSITION(42:44) CHAR(3) NULLIF(41)=X'FF',
" POSITION(46:49) CHAR(4) NULLIF(45)=X'FF',
E" POSITION(51:60) DATE EXTERNAL NULLIF(50)=X'FF',
SITION(62:69) CHAR(8) NULLIF(61)=X'FF',
" POSITION(71:72) SMALLINT NULLIF(70)=X'FF',
SITION(74:74) CHAR(1) NULLIF(73)=X'FF',
TE" POSITION(76:85) DATE EXTERNAL NULLIF(75)=X'FF',
 POSITION(87:91) DECIMAL NULLIF(86)=X'FF',
POSITION(93:97) DECIMAL NULLIF(92)=X'FF',
OSITION(99:103) DECIMAL NULLIF(98)=X'FF')
 WAS SPECIFIED FOR EMPTY TABLESPACE
ASE STATISTICS - NUMBER OF RECORDS=43 FOR TABLE TUTOR01.MY_EMP2
ASE STATISTICS - TOTAL NUMBER OF RECORDS LOADED=43 FOR TABLESPACE

 STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=43
 COMPLETE, ELAPSED TIME=00:00:00
 DSN00016.MYREMP2 PARTITION 1 IS IN COPY PENDING
ION COMPLETE, HIGHEST RETURN CODE=4
******************************** BOTTOM OF DATA ********************************
```

In this case, the table space is in COPY-pending status because the LOAD statement had the LOG NO option. This option indicates that some logging activities are skipped during the LOAD process. Therefore, to ensure that the object is recoverable, Db2 requires an image copy to be taken or some equivalent action. You will resolve this COPY-pending status in the next lesson.

10. Confirm that your new table was populated. Press PF3 to exit. Then, navigate to **SPUFI** and issue a SELECT statement to query the table that you loaded.
    For example, the following query returns the number of rows in the table and confirms that the LOAD utility added rows to the table:

```
 BROWSE    TUTOR01.MYOUTPUT                          Line 00000000 Col 001 080
 Command ===>                                          Scroll ===> PAGE
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
                                                                        00010035
--CREATE TABLE MY_EMP2 LIKE MY_EMP;                                     00060036
  SELECT COUNT (*) FROM MY_EMP2;                                        00061037
---------+---------+---------+---------+---------+---------+---------+---------+

---------+---------+---------+---------+---------+---------+---------+---------+
         42
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
                                                                        00062035
                                                                        00063035
                                                                        00064035
                                                                        00070035
                                                                        00070135
                                                                        00071035
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

11. Determine the database name and table space name of the MY_EMP2 table. Note these values for the next lesson.

    For example, if you have SELECT privilege for the Db2 catalog tables, you might issue the following query to return the database name and table space name of your table:

```
SELECT NAME, CREATOR, DBNAME, TSNAME
  FROM SYSIBM.SYSTABLES
    WHERE NAME='MY_EMP2';
```

    The result contains the database name in the DBNAME column and the table space name in the TSNAME column. If more than one MY_EMP2 table exists, use the CREATOR column to identify the row for the table that you created.

    If you do not have SELECT privilege for the Db2 catalog, ask a colleague who does have SELECT privilege for the Db2 catalog tables to get this information for you by querying the catalog.

What to do next
Press PF3 twice to return to the **DB2I Primary Option Menu** so that you are ready for the next lesson.

Lesson checkpoint
In this lesson, you learned how to complete the following tasks:

- Load data into an empty table by running the LOAD utility
- Check the utility output for any warnings

Lesson 2.3: Running the COPY utility from a JCL job

Writing job control language (JCL) jobs is a common way to run utilities. Although the JCL standards for your company might differ, this tutorial gives you an idea of how the process works.

In the previous lesson, the MY_EMP2 table was left in COPY-pending status. In this tutorial, you will run the COPY utility to resolve this restricted state.

Before you begin
Before you begin

- Complete "Lesson 2.2: Running the LOAD utility from DB2I" on page 361.
- Determine the JCL JOB statement conventions for your site. The JOB statement looks something like the following example:

```
//REPORT  JOB 'TUTOR01',MSGLEVEL=(1,1),CLASS=A,MSGCLASS=A,
//         USER=TUTOR01,PASSWORD=
```

Procedure

Procedure

1. Confirm that the table space is in COPY-pending status by running the DISPLAY DATABASE command.
   a) From the **DB2I Primary Option Menu**, select option 7 DB2 Commands.
   b) In the **DB2 Commands** panel, issue the DISPLAY DATABASE command. Replace the database name (DSN00016 in the following example) with the name of your database that contains the MY_EMP2 table.

```
 F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
F10=Left   F11=Right  F12=Cancel

                              DB2 COMMANDS                    SSID: DB2A
===>

Position cursor on the command line you want to execute and press ENTER

Cmd 1 ===> -DISPLAY DATABASE(DSN00016) SPACENAM(*) RESTRICT
Cmd 2 ===>
...
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

   The results of this command show that your table space is in COPY-pending status:

```
DSNT360I  -DB2A ************************************
DSNT361I  -DB2A *  DISPLAY DATABASE SUMMARY
                *    RESTRICTED
DSNT360I  -DB2A ************************************
DSNT362I  -DB2A     DATABASE = DSN00016  STATUS = RW
                    DBD LENGTH = 4028
DSNT397I  -DB2A
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
MYREMP2  TS    0001 RW,COPY
MYREMP2  TS
******* DISPLAY OF DATABASE DSN00009 ENDED      **********************
DSN9022I  -DB2A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

   The first column (NAME) contains the name of the table space that contains the MY_EMP2 table. In this example, that table space name is MYREMP2. Also, notice that DISPLAY DATABASE output includes abbreviations for restricted states. In this case, COPY means COPY-pending status. For more information about DISPLAY DATABASE output, see DSNT361I (Db2 Messages)
2. Create a member to contain the COPY utility job.
   a) From the **ISPF/PDF Primary Option Menu**, select option 2 Edit.
   b) In the **Edit Entry** panel, type the data set name and the new member name in the **Name** field, and press Enter.

```
  Menu  RefList  RefMode  Utilities  Workstation  Help

                              Edit Entry Panel
Command ===>
                                                                        More:     +
ISPF Library:
   Project . . .
   Group . . . .          . . .          . . .          . . .
   Type  . . . .
   Member  . . .                (Blank or pattern for member selection list)

Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file:
   Name . . . . . 'USER.SRCLIB.TUTORIAL(COPY)'                              +
   Volume Serial . .           (If not cataloged)

Workstation File:
   File Name  . .
                                      Options
Initial Macro  . . . .                   Confirm Cancel/Move/Replace
Profile Name . . . . .                   Mixed Mode
Format Name  . . . . .                   Edit on Workstation
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

The **EDIT** panel opens for the new member.

3. In the **EDIT** panel, type the JCL job for the COPY utility.
You can copy the following example JCL and change it as needed.

```
//COPYTUT JOB 'TUTOR01',MSGLEVEL=(1,1),CLASS=A,
 //         USER=TUTOR01
 //*
 //STEP1 EXEC DSNUPROC,SYSTEM=DB2A,
 //        UID='TUTUTIL001',UTPROC=''
 //SYSIN DD *
   TEMPLATE C1 DSN &JOBNAME..&LOCREM..&PRIBAC.
             DISP (MOD,CATLG,DELETE)
  COPY TABLESPACE  DSN00016.MYREMP2
      COPYDDN (C1,C1) RECOVERYDDN (C1,C1)
 //
```

This example JCL includes the following items:

A JOB statement
The JOB statement gives this job the name COPYTUT, identifies the programmer as TUTOR01, and defines several other parameters. Edit this statement to meet the standards for your organization. For a description of the JOB statement and syntax, see JCL statements: What does the JOB statement do? (z/OS basic skills).

An EXEC statement
The EXEC statement defines the job step and invokes the DSNUPROC procedure to run the utility. DSNUPROC is a standard procedure that is included with Db2 to invoke utilities. By invoking this procedure, you can avoid writing extra JCL. Make sure that you pass the appropriate parameters to this procedure. For example, the SYSTEM parameter must contain your Db2 subsystem name. For a description of the parameters, see The supplied JCL procedure (DSNUPROC) for invoking a Db2 online utility (Db2 Utilities).

TEMPLATE utility control statement
The SYSIN DD statement includes a TEMPLATE utility control statement. The TEMPLATE utility is a Db2 utility that you can use to define templates for any data sets that you need for utility output. Essentially, a template eliminates the need to define JCL DD statements for every data set that you must allocate. In this case, the TEMPLATE utility defines a template for the image copy data sets. The name of the template is C1. The DSN option defines the naming convention for the data sets. You can modify this template to accommodate the data set conventions for your organization. For a description of the TEMPLATE syntax, see Syntax and options of the TEMPLATE control statement (Db2 Utilities).

COPY utility control statement
The COPY utility control statement specifies that the utility will create four image copies: one primary copy and one backup copy at the local site, and one primary copy and one backup copy at

the recovery site. The COPY utility is to use the C1 template to allocate these data sets. In the COPY statement, change the qualified table space name to the name of your table space that is in COPY-pending status. Notice that the COPY utility, like many utilities, operates on the table space. For help with the COPY statement syntax, see Syntax and options of the COPY control statement (Db2 Utilities).

Remember: All of your JCL statements (the ones that start with //) must start in the first column. JCL also has rules about which text can go in certain columns. For help with JCL, see Basic JCL concepts (z/OS Basic Skills).

4. Submit the job to run the utility.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       USER.SRCLIB.TUTORIAL(COPY) - 01.00              Columns 00001 00072
Command ===> SUB                                           Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000200 //COPYTUT JOB 'TUTOR01',MSGLEVEL=(1,1),CLASS=A,
000300 //         USER=TUTOR01,PASSWORD=C0DESHOP
000310 //*
000600 //STEP1 EXEC DSNUPROC,SYSTEM=DB2A,
000610 //        UID='TUTUTIL001',UTPROC=''
000700 //SYSIN DD *
001000 TEMPLATE C1 DSN &JOBNAME..&LOCREM..&PRIBAC.
001100             DISP (MOD,CATLG,DELETE)
001200 COPY TABLESPACE  DSN00016.MYREMP2
001300     COPYDDN (C1,C1) RECOVERYDDN (C1,C1)
001400 //
****** **************************** Bottom of Data ****************************
```

If you receive the message `ENTER JOBNAME CHARACTER(S) -`, type any character and press Enter.

A message indicates that the job was submitted and provides the job number.

```
JOB COPYTUT(JOB00174) SUBMITTED
***
```

5. View the job output.
   a) Issue the `TSO SDSF ST` command to display the job output in the status of jobs log.
   b) Find your job in the list and issue the S line command next to it to display the utility output.
   c) Determine if the utility completed successfully.
      - A utility return code of DSNUTILB 0000 indicates that the utility completed successfully.
      - Any utility return code other than DSNUTILB 0000 indicates that the utility did not complete successfully. Use the Db2 utility messages to debug the error. Db2 utility messages begin with DSNU. To see these messages, type `BOTTOM` in the **Command** field, and press Enter. Press PF11 to scroll right.
      - If you see JCL errors, use the following resources to help correct the errors:
        – Basic JCL concepts (z/OS Basic Skills)
        – JCL statement fields (MVS JCL Reference)

      To correct any errors in your JCL or to debug any other problems, press PF3 multiple times to return to the data set member that contains your JCL. Correct the errors and then repeat steps 4 and 5 until the job completes successfully.
6. View the image copies:
   a) Press PF3 until you return to the **ISPF/PDF Primary Option Menu**.
   b) Type option `3.4` to open the **DSLIST** utility.
   c) In the **Data Set List Utility** panel, type `COPYTUT` in the **Dsname level** field, and press Enter.

      Your image copy data sets begin with the qualifier COPYTUT because of the template that you used. The template specified that your job name (indicated by the variable `&JOBNAME.`) will be the first qualifier in the data set name. In this case, your job name was COPYTUT. For a complete

description of template names and variables, see Syntax and options of the TEMPLATE control statement (Db2 Utilities).

The data sets that contain the image copies are listed.

```
  Menu  Options  View  Utilities  Compilers  Help

DSLIST - Data Sets Matching COPYTUT                                 Row 1 of 4
Command ===>                                                     Scroll ===> PAGE

Command - Enter "/" to select action                 Message           Volume
-------------------------------------------------------------------------------
         COPYTUT.L.B                                                   SCR03
         COPYTUT.L.P                                                   SCR03
         COPYTUT.R.B                                                   SCR03
         COPYTUT.R.P                                                   SCR03
****************************** End of Data Set list ****************************

 F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
F10=Left   F11=Right  F12=Cancel
```

If you view the contents of the data sets, you might notice that the data is not in an easily readable format. The image copies are not written for people to read, but for Db2 to use for recovery.

7. Verify that the COPY-pending status was resolved for the table space that contains the MY_EMP2 table. In this example, the table space name is MYREMP2.
   a) From the **DB2I Primary Option Menu**, type option 7 Commands.
   b) Issue the following DISPLAY DATABASE command. Remember to change the database name (DSN00016) and table space name (MYREMP2) to your values.

```
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel

                              DB2 COMMANDS                   SSID: DB2A
 ===>

 Position cursor on the command line you want to execute and press ENTER

 Cmd 1 ===> -DISPLAY DATABASE(DSN00016) SPACENAM(MYREMP2)
 Cmd 2 ===>
...
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

The following output is displayed. The status of the table space is read/write only, which means that the table space is no longer in a restrictive state.

```
DSNT360I  -DB2A ************************************
DSNT361I  -DB2A *  DISPLAY DATABASE SUMMARY
                *    GLOBAL
DSNT360I  -DB2A ************************************
DSNT362I  -DB2A     DATABASE = DSN00016  STATUS = RW
                  DBD LENGTH = 4028
DSNT397I  -DB2A
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
MYREMP2  TS    0001 RW
MYREMP2  TS
******* DISPLAY OF DATABASE DSN00016 ENDED      **********************
DSN9022I  -DB2A DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION
***
```

Lesson checkpoint

In this lesson, you learned how to complete the following tasks:

- Run a utility from a JCL job
- Create a template for your data sets by using the TEMPLATE utility
- Check the status of a table space by issuing a DISPLAY DATABASE command

Chapter 11. New-user information roadmap

This roadmap outlines the information resources that are available for users who are new to Db2 for z/OS. These resources provide information about various subject areas, such as learning basic skills, administration, programming, and troubleshooting for Db2 for z/OS.

Tip: This roadmap identifies information resources that are most appropriate for getting started with Db2 for z/OS. If you are an experienced Db2 for z/OS administrator, see the administration resources in Chapter 12, “Db2 11 for z/OS information roadmap,” on page 375.

If you are an experienced application developer, see the programming resources in Chapter 13, “Application Developer information roadmap,” on page 381.

Product overview

z/OS Basic Skills education
New to z/OS? You've come to the right place! The z/OS Basic Skills education offers the fastest way to learn and become productive onz/OS.

Introduction to the New Mainframe: z/OS Basics
This IBM Redbooks® publication provides you with the background knowledge and skills necessary to begin using the basic facilities of a mainframe computer. This book is designed to introduce new users to mainframe concepts and to help new users prepare for a career in large systems computing.

IBM Mainframe: Make the Extraordinary Possible
This video explains how the mainframe is essential in today's digital world, enabling 90% of Fortune 500 companies.

zIQ by IBM Z
This series of videos explains the relevance and power of IBM Z.

Db2 for z/OS training paths
Looking for educational courses about Db2 for z/OS? Visit this website to view recommended training paths and training options by topic area, such as administration, application programming, database design, IBM Db2 Tools for z/OS, and more. Training opportunities are available online and onsite.

IBM IBM Z Course Materials for the Community
This web page contains course materials to assist the mainframe community with skills. Most of the materials are intended for use in a 15-week semester IT course; however, some of these resources are smaller and can be used independently.

Db2 for z/OS Product Documentation
This technote provides links to the Db2 for z/OS product information in PDF format, along with links to Db2 for z/OS and related Program Directories and other valuable information resources.

Planning

Db2 11 Technical Overview
This IBM Redbooks publication introduces the enhancements made available with this version of Db2 for z/OS. The contents help you understand the new functions and performance enhancements, start planning for exploiting the key new capabilities, and justify the investment in installing or migrating.

Installing and migrating

Installing and migrating Db2 (Db2 Installation and Migration)
This information provides an overview of the essential steps involved in installing or migrating to a new release of Db2 for z/OS.

Program Directories for Db2 for z/OS
This web page links to Program Directories for Db2 for z/OS and related components. Program Directories provide helpful information that you need when you install or migrate to a new version of a product.

© Copyright IBM Corp. 2001, 2020

Administering

Certifications for Db2 administrators

Db2 for z/OS Database Administration Core training path
Use this training path to see the courses that you need to take for Db2 for z/OS database administration.

IBM Professional Certification Program
This web page provides you with information about Db2 certification. This information applies to Db2 products on all of the different operating systems, including z/OS.

Db2 Utilities

Introduction to Db2 utilities
Db2 utilities help maintain data in your Db2 for z/OS databases. Generally, you use Db2 utilities to perform large-scale operations across one or more table spaces.

Db2 utilities (Db2 Utilities)
Reference information and usage guidance for online and stand-alone Db2 utilities.

Db2 for z/OS Utilities in Practice
This IBM Redpaper™ publication helps Db2 database administrators, Db2 system programmers, and anyone who runs Db2 for z/OS utilities implement best practices.

Tutorial: Running Db2 utilities
You can use several different methods to run Db2 utilities. This module shows you two of these methods: using a Db2 Interactive (DB2I) panel and writing job control language (JCL) jobs.

Db2 utilities packaging (Db2 Utilities)
Several utilities are included with Db2 for z/OS at no extra charge. Other utilities are available in the separate Db2 Utilities Suite for z/OS product.

Administration with IBM Db2 Tools for z/OS

Db2 Tools for z/OS
This web page provides you with information about the various tools that can help you manage database administration and change management processes.

Db2 Administration Solution Pack for z/OS: Streamlining Db2 for z/OS Database Administration
This IBM Redbooks publication discusses how the products in the Db2 Administration Solution Pack for z/OS can help database administrators more efficiently complete tasks that are associated with object management, change management, application management, and configuration management.

Security

Db2 security: An interactive model
This interactive model provides an overview of key Db2 security components and processes. Hover over the objects to explore and learn more about the basics of Db2 security in the z/OS environment.

Db2 stored procedures

Implementing Db2 stored procedures
A stored procedure is executable code that can be called by other programs. You might choose to use stored procedures for code that is used repeatedly. This information describes some benefits of using stored procedures and how to create and call stored procedures.

Scenarios

Scenarios for using Db2
These scenarios illustrate how some organizations (an international bank, a multi-campus university system, and an electric company) might successfully use Db2 for z/OS.

Samples

Db2 programming samples (Db2 Programming samples)

You can find application programming samples that you can use to get started creating applications that access data in Db2 for z/OS. The collection of samples come from various sources, including:

- Content that is adapted from existing IBM developerWorks® and other articles from around the web.
- Samples that were previously delivered throughout in the existing Db2 programming information.
- Examples that were previously supplied with Db2 in the DSN1110.SDSNSAMP library, but only recently included in their entirety in the documentation.

Db2 for z/OS developerWorks Exchange

This forum enables you to exchange sample programs, SQL statements, utility jobs, and more with your colleagues.

IDUG Codeplace

On this section of the International Db2 Users Group (IDUG) website, you can find scripts, code fragments, and samples for Db2 technology. You can also submit your own samples.

Getting started

Db2 basics tutorial: Working with Db2 for z/OS interactively

This tutorial shows you how to use Db2 for z/OS in the z/OS terminal interface to complete some basic database administration tasks.

Tutorial: Using the command line processor

This tutorial demonstrates how you can interactively query and modify Db2 data, call stored procedures, and issue z/OS UNIX System Services commands through the command line processor.

Developing

Application programming for Db2

You use application programs that contain SQL statements to perform data definition or data manipulation operations on Db2 objects. This information guides you through the basic concepts that you need to know before you start writing SQL applications.

Db2 for z/OS Best Practices

Are you looking for recommendations and tips on how to best use Db2 for z/OS to meet your business needs? This website provides best practices recommendations on a variety of subjects, including developing application programs, from some of your favorite Db2 experts based on field experience. You can download and watch presentations, listen to podcasts and previously recorded webcasts, and read articles.

Db2 and Java: The Big Picture

This developerWorks article provides an introduction to Java support in Db2 for z/OS.

Processing XML data with Db2 pureXML

Learn how you can use pureXML with your client applications to manage XML data in Db2 tables. You can store well-formed XML documents in their hierarchical form and retrieve all or portions of those documents.

Tutorial for pureXML

This tutorial demonstrates how to set up a Db2 database system to store XML data and to perform basic operations with XML data.

Managing performance

Db2 performance management

Managing the performance of a Db2 subsystem involves understanding a wide range of system components. This information helps you understand the performance of those components, how to monitor the components, and how to identify problem areas.

Db2 11 Performance Topics
This IBM Redbooks publication provides an overview of the performance improvements in Db2 11. Performance improvements in Db2 11 continue to focus on CPU savings and enhancements through innovations within Db2 and the synergy with IBM Z hardware and software.

Troubleshooting and support

Code fixes

IBM Support Portal: Db2 for z/OS
Use the IBM Support Portal for Db2 for z/OS to search for both PTFs and APARs. This website also provides links to common troubleshooting topics.

Db2 for z/OS keywords for searching APAR text
This document contains keywords that you can use to optimize your results when searching APAR text.

General IBM Software Support information

Introduction to troubleshooting problems in Db2 for z/OS
If you are new to Db2 for z/OS, you should learn about the tools and techniques that are available to help you troubleshoot and resolve problems.

Software Support Handbook
This manual provides guidelines and reference materials that you might need when you require help from IBM Software Support.

APAR Closing Codes and Acronyms
This section of the *Software Support Handbook* provides lists of acronyms and terms that are used frequently by IBM Support when solving customer problems.

Reference

Information Management software lifecycle dates
The Information Management software lifecycle page shows the dates that marketing and support services are available for each version of Db2 for z/OS, along with the end-of-service (EOS) dates.

Community resources

This section may provide links or references to non-IBM websites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM websites or third-party resources (including any Lenovo website) that may be referenced, accessible from, or linked to any IBM site. A link to a non-IBM web site does not mean that IBM endorses the content or use of such website or its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products, or other materials on or available from those sites or resources. When you access a non-IBM website, even one that may contain the IBM-logo, please understand that it is independent from IBM, and that IBM does not control the content on that website. It is up to you to take precautions to protect yourself from viruses, worms, trojan horses, and other potentially destructive programs, and to protect your information as you deem appropriate.

Forums and organizations

IBM Information Management Communities
This website provides convenient links to a variety of user groups and forums, where the Information Management user community can connect, discuss, and learn from each others' experiences using IBM Information Management software solutions.

International Db2 Users Group (IDUG)
This user group provides a forum for the Db2 community to learn and share experiences about effectively using the Db2 family of products, including Db2 for z/OS, Db2 for Linux, UNIX, and Windows, DB2 for i, DB2 for VSE & VM, and others.

SHARE
This community provides Information Technology professionals with access to continuous education and training, and valuable professional networking, helping you to achieve your business goals.

Blogs

Db2 for z/OS news from the lab blog
News about Db2 for z/OS from the IBMers who design, build, test, and support Db2.

Getting the most out of Db2 for z/OS
This blog by Willie Favero, a certified IT software specialist for Db2 for z/OS, offers hints and tips about Db2 for z/OS and IBM Z, including tips on migration, performance, and maintenance.

DB2Portal blog
This blog by Craig Mullins, a data management strategist and consultant, offers news, views, and items of interest on Db2 database management and mainframe systems.

Social media

The World of Db2 for z/OS
This networking site provides a variety of resources about Db2, including videos, blogs, podcasts, and discussions.

Follow @Db2LabNews
News about Db2 for z/OS from the IBMers who design, build, test, and support Db2.

Follow @Db2world
The latest Db2, Analytics, Machine Learning, and Cloud news.

The World of Db2 for z/OS
This networking site provides a variety of resources about Db2, including videos, blogs, podcasts, and discussions.

Db2 for z/OS - Stay In Touch!
Be a part of the Db2 for z/OS community on LinkedIn. This group communicates to our customers and business partners what is happening in the world of Db2 for z/OS.

Db2 Facebook community
This site connects members of the Db2 community. On this Facebook page, you can find links to blogs and videos about Db2, and become a fan of Db2 on Facebook.

Chapter 12. Db2 11 for z/OS information roadmap

This roadmap outlines the information resources that are available for Db2 for z/OS administrators. These resources provide information about various subject areas, such as installation, administration, and troubleshooting for Db2 for z/OS.

Tip: This roadmap identifies information resources that are most appropriate for experienced Db2 for z/OS administrators. If you are just getting started with Db2 for z/OS, see Chapter 11, "New-user information roadmap," on page 369.

If you are an experienced application developer, see the programming resources in Chapter 13, "Application Developer information roadmap," on page 381.

Product overview

IBM Db2 for z/OS: The Database for Critical Transactions and Analytics
This web page describes the features and benefits of Db2 for z/OS.

Db2 for z/OS Product Documentation
This technote provides links to the Db2 for z/OS product information in PDF format, along with links to Db2 for z/OS and related Program Directories and other valuable information resources.

Mainframe success stories
Read what our clients are saying about the power and flexibility of IBM Z.

Planning

General planning

Release announcement for Db2 11 for z/OS
This announcement letter summarizes the key enhancements that are available in this release of Db2 for z/OS.

Db2 11 for z/OS Technical Overview
This IBM Redbooks publication introduces the enhancements made available Db2 11. The contents help you understand the new functions and performance enhancements, start planning for exploiting the key new capabilities, and justify the investment in installing or migrating.

Planning for specific functions

Preparing for Db2 data sharing
This information provides details about planning for Db2 data sharing, establishing naming conventions, estimating storage requirements, and so on.

Db2 utilities packaging (Db2 Utilities)
Several utilities are included with Db2 for z/OS at no extra charge. Other utilities are available in the separate Db2 Utilities Suite for z/OS product.

Installing and migrating

Preparing your system to install or migrate Db2
This information describes estimating Db2 storage needs, loading Db2 libraries, installing Db2, migrating to conversion mode, enabling new-function mode, verifying your installation or migration with the sample applications, and connecting the IMS attachment facility.

Program Directories for Db2 for z/OS
This web page links to Program Directories for Db2 for z/OS and related components. Program Directories provide helpful information that you need when you install or migrate to a new version of a product.

© Copyright IBM Corp. 2001, 2020

Configuring

Configuring TLS/SSL for Secure Client-Server Communications
This IBM Redpaper provides information about how to set up and configure Db2 for z/OS with Transport Layer Security (TLS), the modern version of Secure Sockets Layer (SSL).

System z Parallel Sysplex Best Practices
This IBM Redbooks publication provides information about the best way to design, implement, and manage an IBM Z Parallel Sysplex to deliver the levels of performance and availability that are required by your organization.

Administering

Certifications for Db2 administrators

Db2 for z/OS Database Administration Core training path
Use this training path to see the courses that you need to take for Db2 for z/OS database administration.

IBM Professional Certification Program
This web page provides you with information about Db2 certification. This information applies to Db2 products on all of the different operating systems, including z/OS.

Db2 utilities

Introduction to Db2 utilities
Db2 utilities help maintain data in your Db2 for z/OS databases. Generally, you use Db2 utilities to perform large-scale operations across one or more table spaces.

Db2 utilities (Db2 Utilities)
Reference information and usage guidance for online and stand-alone Db2 utilities.

Db2 utilities packaging (Db2 Utilities)
Several utilities are included with Db2 for z/OS at no extra charge. Other utilities are available in the separate Db2 Utilities Suite for z/OS product.

Db2 for z/OS Utilities in Practice
This IBM Redpaper™ publication helps Db2 database administrators, Db2 system programmers, and anyone who runs Db2 for z/OS utilities implement best practices.

Db2 for z/OS Utilities Development Blog
A community hints and tips blog for Db2 utilities.

Administration with Db2 Tools

Db2 Tools for z/OS
This web page provides you with information about the various tools that can simplify management, maximize performance, and lower operating costs.

Db2 Administration Tool for z/OS
This web page provides general information and resources for the Db2 Administration Tool for z/OS, which provides a comprehensive set of functions that help Db2 personnel manage their Db2 environments efficiently and effectively.

Managing Db2 10 for z/OS Using the IBM Db2 Administration Tool for z/OS Version 10
This IBM Redbooks publication highlights the data administration enhancements introduced by Db2 Administration Tool for z/OS Version 10, showing scenarios of their use with the new functions provided by DB2 10. Although written in particular for DB2 10, this publication is also helpful as you administer later releases of Db2 for z/OS.

Db2 Administration Solution Pack for z/OS: Streamlining Db2 for z/OS Database Administration
This IBM Redbooks publication discusses how the products in the Db2 Administration Solution Pack for z/OS can help database administrators more efficiently complete tasks that are associated with object management, change management, application management, and configuration management.

Managing Db2 for z/OS Utilities with Db2 Tools Solution Packs
This IBM Redbooks publication highlights productivity and performance gains for database administrators when using the IBM Db2 Utilities Solution Pack and the IBM Db2 Fast Copy Solution

Pack. This publication explains the functions of the tools in these solution packs, and provides real-life scenarios and best practices for adopting utilities.

Security

Db2 security: An interactive model
This interactive model provides an overview of key Db2 security components and processes. Hover over the objects to explore and learn more about the basics of Db2 security in the z/OS environment.

Security Functions of IBM Db2 10 for z/OS
This IBM Redbooks publication highlights added functions in the areas of security, regulatory compliance, and audit capability. Although written for DB2 10, this information is helpful when managing security in later releases of Db2 for z/OS.

Db2 stored procedures

Implementing Db2 stored procedures
A stored procedure is executable code that can be called by other programs. You might choose to use stored procedures for code that is used repeatedly. This information describes some benefits of using stored procedures and how to create and call stored procedures.

Data Studio and Db2 for z/OS Stored Procedures
This IBM Redpaper publication discusses how you can use tools, in particular IBM Data Studio, to accelerate the process of developing and debugging stored procedures. This paper is a major update to Part 6, "Cool tools for an easier life," of the IBM Redbooks publication *Db2 9 for z/OS Stored Procedures: Through the CALL and Beyond*.

Db2 for z/OS Stored Procedures: Through the CALL and Beyond (IBM Redbooks)
Although this IBM Redbooks publication was written for DB2 9, this information is also helpful as you design, install, manage, and tune stored procedures with later releases of Db2 for z/OS.

Scenarios

Case studies: BMW Group develops eco-friendly innovation for smart drivers with IBM
This case study shares the success that BMW Group had migrating to and using Db2 11 as part of the Early Support Program (ESP).

Case studies: JN Data gets the early-adopter advantage for its growing business
This case study shares the success that JN Data has had as an early-adopter of Db2 11.

Samples

Db2 programming samples (Db2 Programming samples)
You can find application programming samples that you can use to get started creating applications that access data in Db2 for z/OS. The collection of samples come from various sources, including:

- Content that is adapted from existing IBM developerWorks and other articles from around the web.
- Samples that were previously delivered throughout in the existing Db2 programming information.
- Examples that were previously supplied with Db2 in the DSN1110.SDSNSAMP library, but only recently included in their entirety in the documentation.

Db2 for z/OS developerWorks Exchange
This forum enables you to exchange sample programs, SQL statements, utility jobs, and more with your colleagues.

IDUG Codeplace
On this section of the International Db2 Users Group (IDUG) website, you can find scripts, code fragments, and samples for Db2 technology. You can also submit your own samples.

Managing performance

General information

Db2 11 Performance Topics

This IBM Redbooks publication provides an overview of the performance improvements in Db2 11. Performance improvements in Db2 11 focus on CPU savings and enhancements through innovations within Db2 and the synergy with IBM Z hardware and software.

Subsystem and Transaction Monitoring and Tuning with Db2 11

This IBM Redbooks publication provides a detailed discussion of the facilities of Db2 for z/OS, which allow complete monitoring of a Db2 environment. This book discusses the use of the Db2 instrumentation facility component (IFC) to provide monitoring of Db2 data and events, and includes suggestions for related tuning.

IBM Db2 11 for z/OS: Buffer Pool Monitoring and Tuning

This IBM Redpaper publication provides valuable information about monitoring and tuning buffer pools. Although written specifically for Db2 11, this information can also be applied to DB2 10.

The value of IBM Db2 for z/OS hash access for OLTP transactions

This white paper provides an overview of how you can use the new hash organization in DB2 10 to improve response times and reduce the cost of queries that retrieve single rows from a table containing unique keys. This information helps you to decide whether your environment can benefit from this new capability.

Using performance tools

Enabling Db2 to work with IBM Db2 Analytics Accelerator for z/OS

This information discusses how you can use IBM Db2 Analytics Accelerator for z/OS, or an equivalent accelerator, to run queries on behalf of Db2. For certain types of queries, such as business intelligence (BI) queries, processing on an accelerator server can be faster than processing on Db2.

Reliability and Performance with IBM Db2 Analytics Accelerator for z/OS Version 4.1

This IBM Redbooks publication helps technical decision makers understand the benefits of IBM Db2 Analytics Accelerator for z/OS, Version 4.1 when used with Db2 11.

Hybrid Analytics Solution using IBM Db2 Analytics Accelerator for z/OS V3.1

This IBM Redbooks publication helps technical decision makers understand the benefits and major new functions of IBM Db2 Analytics Accelerator for z/OS, Version 3.1.

Optimizing Db2 Queries with IBM Db2 Analytics Accelerator for z/OS

This IBM Redbooks publication evaluates the potential benefits of using IBM Db2 Analytics Accelerator for z/OS, a high-performance accelerator appliance, to optimize the processing of certain Db2 queries. This publication defines a business analytics scenario, describes the installation and integration steps with the Db2 environment, and evaluates performance improvements.

Db2 Query Monitor for z/OS information

This information supports Db2 Query Monitor, which gives you the ability to efficiently customize and tune your SQL workload and Db2 objects.

Optim Query Workload Tuner for Db2 for z/OS

Db2 Query Workload Tuner for z/OS helps developers create efficient queries and build tuning skills. Db2 Query Workload Tuner for z/OS cuts cost and improves performance by providing expert advice on writing high-quality queries and improving database design. Its easy-to-use advisors can help developers to write more efficient SQL queries.

Data Server Manager

With Data Server Manager, DBAs and application developers can format SQL statements so that they are easier to read, generate visual representations of access plans, and get recommendations for collecting statistics or improving indexes. Access plan graphs show where statements are using the most time and show where tables are being scanned. Tuning advisors are also available to provide expert recommendations about statistics, indexes, and performance improvement.

Troubleshooting and support

Code fixes

IBM Support Portal: Db2 for z/OS
Use the IBM Support Portal for Db2 for z/OS to search for both PTFs and APARs. This website also provides links to common troubleshooting topics.

Db2 for z/OS keywords for searching APAR text
This document contains keywords that you can use to optimize your results when searching APAR text.

General IBM Support information

Introduction to troubleshooting problems in Db2 for z/OS
If you are new to Db2 for z/OS, you should learn about the tools and techniques that are available to help you troubleshoot and resolve problems.

How to obtain a copy of Db2 Diagnosis Guide and Reference
This technote provides information about how to obtain a copy of Db2 Diagnosis Guide and Reference. This book provides information for diagnosing Db2 problems.

Software Support Handbook
This manual provides guidelines and reference materials that you might need when you require help from IBM Support.

APAR Closing Codes and Acronyms
This section of the *Software Support Handbook* provides lists of acronyms and terms that are used frequently by IBM Support when solving customer problems.

Reference

Information Management software lifecycle dates
The Information Management software lifecycle page shows the dates that marketing and support services are available for each version of Db2 for z/OS, along with the end-of-service (EOS) dates.

Community resources

This section may provide links or references to non-IBM websites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM websites or third-party resources (including any Lenovo website) that may be referenced, accessible from, or linked to any IBM site. A link to a non-IBM web site does not mean that IBM endorses the content or use of such website or its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products, or other materials on or available from those sites or resources. When you access a non-IBM website, even one that may contain the IBM-logo, please understand that it is independent from IBM, and that IBM does not control the content on that website. It is up to you to take precautions to protect yourself from viruses, worms, trojan horses, and other potentially destructive programs, and to protect your information as you deem appropriate.

Forums and organizations

IBM Information Management Communities
This website provides convenient links to a variety of user groups and forums, where the Information Management user community can connect, discuss, and learn from each others' experiences using IBM Information Management software solutions.

International Db2 Users Group (IDUG)
This user group provides a forum for the Db2 community to learn and share experiences about effectively using the Db2 family of products, including Db2 for z/OS, Db2 for Linux, UNIX, and Windows, DB2 for i, Db2 Express, Db2 Everywhere, and Db2 Data Warehouse Edition.

SHARE
This community provides Information Technology professionals with access to continuous education and training, and valuable professional networking, helping you to achieve your business goals.

Blogs

Db2 for z/OS news from the lab blog
News about Db2 for z/OS from the IBMers who design, build, test, and support Db2.

Getting the most out of Db2 12
This blog by Willie Favero, a certified IT software specialist for Db2 for z/OS, offers hints and tips about Db2 for z/OS and IBM Z, including tips on migration, performance, and maintenance.

DB2utor
This blog by Troy Coleman provides hints, tips, and best practices for using Db2 for z/OS and related products.

Robert's Db2 blog
In this blog Robert Catterall, an IBM Db2 specialist, writes about Db2 for z/OS, providing information about topics such as performance tuning, data warehousing, data sharing, service-oriented architecture (SOA), and more.

DB2Portal blog
This blog by Craig Mullins, a data management strategist and consultant, offers news, views, and items of interest on Db2 database management and mainframe systems.

Db2 for z/OS Utilities Development Blog
A community hints and tips blog for Db2 utilities.

DB2usa: Blog about Db2 for z/OS
Visit this blog site each week to find the latest technical information that is available on the web about Db2 for z/OS.

Follow Db2 for z/OS

Follow @Db2LabNews
News about Db2 for z/OS from the IBMers who design, build, test, and support Db2.

Follow @Db2world
The latest Db2, Analytics, Machine Learning, and Cloud news.

The World of Db2 for z/OS
This networking site provides a variety of resources about Db2, including videos, blogs, podcasts, and discussions.

Db2 for z/OS - Stay In Touch!
Be a part of the Db2 for z/OS community on LinkedIn. This group communicates to our customers and business partners what is happening in the world of Db2 for z/OS.

Db2 Facebook community
This site connects members of the Db2 community. On this Facebook page, you can find links to blogs and videos about Db2, and become a fan of Db2 on Facebook.

Chapter 13. Application Developer information roadmap

This roadmap outlines the information resources that are available for application developers who are creating applications that access data in Db2 for z/OS.

Tip: This roadmap identifies information resources that are most appropriate for experienced Db2 for z/OS programmers. If you are just getting started with Db2 for z/OS, see Chapter 11, "New-user information roadmap," on page 369.

If you are an experienced administrator, see the programming resources in Chapter 12, "Db2 11 for z/OS information roadmap," on page 375.

Programming samples

Db2 programming samples (Db2 Programming samples)
You can find application programming samples that you can use to get started creating applications that access data in Db2 for z/OS. The collection of samples come from various sources, including:

- Content that is adapted from existing IBM developerWorks and other articles from around the web.
- Samples that were previously delivered throughout in the existing Db2 programming information.
- Examples that were previously supplied with Db2 in the DSN1110.SDSNSAMP library, but only recently included in their entirety in the documentation.

General Db2 database programming

Planning for and designing Db2 applications
Detailed information about planning and designing applications that connect to Db2, and read, add or modify data, and manipulate Db2 objects.

Writing Db2 applications
Detailed information about writing applications that connect to Db2, and read, add or modify data, and manipulate Db2 objects.

"Tools and IDEs for developing Db2 applications" on page 233
Application developers can use various application development tools and integrated development environments (IDEs) to create database applications, and stored procedures that access and use data in Db2 for z/OS.

Db2 for z/OS Best Practices
Are you looking for recommendations and tips on how to best use Db2 for z/OS to meet your business needs? This website provides best practices recommendations on a variety of subjects, including developing application programs, from some of your favorite Db2 experts based on field experience. You can download and watch presentations, listen to podcasts and previously recorded webcasts, and read articles.

Db2 Query Management Facility
This web page describes the benefits of Db2 Query Management Facility for z/OS, which provides a world-class set of features and functions for Db2 queries, reporting, dashboarding and analytics.

Implementing Db2 stored procedures
Detailed information about developing executable code that can be called by other programs.

Db2 for z/OS Stored Procedures: Through the CALL and Beyond
This IBM Redbooks publication helps you design, install, manage, and tune stored procedures in Db2 for z/OS.

Db2 for z/OS: Dynamic Statement Cache
This white paper provides a detailed overview of the dynamic statement cache.

Analytics application development

© Copyright IBM Corp. 2001, 2020

Integrate Db2 for z/OS with InfoSphere BigInsights, Part 1: Set up the InfoSphere BigInsights connector for Db2 for z/OS (IBM developerWorks)

Learn how to set up integration between IBM Db2 for z/OS and IBM InfoSphere BigInsights®. This article is the first of a two-part series that describes how to create a big data solution by integrating IBM Db2 for z/OS and IBM InfoSphere BigInsights. In Part 1, learn how to configure the InfoSphere BigInsights connector in Db2 for z/OS.

Integrate Db2 for z/OS with InfoSphere BigInsights, Part 2: Use the InfoSphere BigInsights connector to perform analysis using Jaql and SQL (IBM developerWorks)

Learn how to set up integration between IBM Db2 for z/OS and IBM InfoSphere BigInsights™. In this article, explore how to create a big data solution that integrates IBM Db2 for z/OS and IBM InfoSphere BigInsights Learn how to use the InfoSphere BigInsights connector in Db2 for z/OS to perform analysis by using Jaql and SQL.

Predictive analytics using IBM SPSS Modeler in Db2 for z/OS (IBM developerWorks)

A tutorial focused on predictive analytics using IBM SPSS® Modeler and data stored in Db2 for z/OS. It illustrates how to use the SPSS Modeler workbench to create predictive models with in-database mining, SQL push-back, and UDF scoring. It walks through the steps for integrating real-time scoring for Db2 for z/OS into an OLTP application. It also shows what needs to be done in the Db2 server and the information an application developer needs to know to create an enterprise solution for in-database transactional scoring and batch scoring.

Connectivity and distributed database development

Db2 Connect

This web page contains information about Db2 Connect, a tool that makes your company's host data directly available to your personal computer and LAN-based workstations.

DB2 9 for z/OS: Distributed Functions

This IBM Redbooks publication describes how to set up a DDF environment, how to deploy DDF capabilities in different configurations, and how to develop applications that access distributed databases. Although this publication was written for DB2 9, the information can be useful for customers on later releases of Db2 for z/OS.

Mobile application development

Build a Db2 for z/OS mobile application using IBM MobileFirst (IBM developerWorks)

Learn how you can build a Db2 for z/OS mobile application using IBM MobileFirst® Developer Edition. Using an easy-to-adapt example, this tutorial shows what needs to be done from the front end to the back-end database server to make this happen.

Build a Db2 for z/OS mobile application using Db2 Adapter for z/OS Connect, Part 1: Expose Db2 functionality (IBM developerWorks)

To create effective, enterprise-ready mobile applications, you need to make the back-end data easier to consume. APIs in IBM Db2 Adapter for z/OS expose Db2 for z/OS functionality to applications. See an easy-to adapt example to learn how you can build a Db2 for z/OS mobile application using Db2 Adapter for z/OS Connect and IBM MobileFirst Developer Edition.

Build a Db2 for z/OS mobile application using Db2 Adapter for z/OS Connectart 2: Consume a Db2 Adapter service

To create effective, enterprise-ready mobile applications, you need to make the back-end data easier to consume. APIs in IBM Db2 Adapter for z/OS expose Db2 for z/OS functionality to applications. In Part 2 of the series, learn how to build a mobile application using IBM MobileFirst Developer Edition by consuming a Db2 Adapter service.

Java, JDBC, and J2EE application development

Java application development for IBM data servers

Detailed information about writing and preparing Java programs that access Db2 for z/OS databases.

Db2 and Java: The Big Picture

This developerWorks article provides an introduction to Java support in Db2 for z/OS.

Db2 for z/OS and WebSphere Integration for Enterprise Java Applications

This IBM Redbooks publication explains how you can use IBM WebSphere Application Server to deploy, integrate, execute, and manage Java applications.

Managing Enterprise Java Applications for Db2

This paper helps customers who want more control over their Java applications understand the full extent of what is offered to them byDb2 and describes the features that developers should use to maximize the manageability of their applications.

Lab: z/OS Db2 access using Java

A lab that describes how to connect to a Db2 running on z/OS and how to run SQL statements with Java.

XML application development

Extremely pureXML in DB2 10 for z/OS

This IBM Redbooks publication documents the steps for the implementation of a simple but meaningful XML application scenario. Although written for DB2 10, this information is also useful for later releases of Db2 for z/OS.

XML or JSON: Guidelines for what to choose for Db2 for z/OS (IBM developerWorks)

IBM Db2 for z/OS offers document storage support for both JSON and XML. It is not always apparent whether JSON or XML is most suitable for a particular application. This article provides guidelines to help you select XML or JSON. It includes examples of creating, querying, updating, and managing in both JSON and XML in Db2 for z/OS.

JSON application programming

Use a SQL interface to handle JSON data inDb2 11 for z/OS (IBM developerWorks)

This tutorial focuses on a SQL interface recently introduced in Db2 11 that allows extraction and retrieval of JSON data from BSON objects and conversion from JSON to BSON. With this new feature, users can manage JSON data without relying on Db2 NoSQL JSON APIs. Instead, SQL interfaces can be used for JSON manipulation. Learn about the setup/configuration and get illustrations for common JSON usage inside Db2 11. Hints and tips are provided to improve performance and prevent potential pitfalls.

XML or JSON: Guidelines for what to choose for Db2 for z/OS (IBM developerWorks)

IBM® Db2 for z/OS offers document storage support for both JSON and XML. It is not always apparent whether JSON or XML is most suitable for a particular application. This article provides guidelines to help you select XML or JSON. It includes examples of creating, querying, updating, and managing in both JSON and XML in Db2 for z/OS.

Application deployment

Deploy database applications against Db2 for z/OS with IBM UrbanCode deploy

A tutorial that describes how to define a deployment process against various environments, including multiple tenant environments and multiple sysplex environments with the delivered plug-ins in IBM UrbanCode® Deploy.

IBM Db2 for z/OS Package (UrbanCode Deploy)

Importable sample template that describe how to define a deployment process against various environments, including multiple tenant environments and multiple sysplex environments, by using the plug-ins that are included with IBM UrbanCode Deploy.

UrbanCode Deploy

IBM UrbanCode Deploy is a tool for automating application deployments through your environments. It is designed to facilitate rapid feedback and continuous delivery in agile development while providing the audit trails, versioning and approvals needed in production.

Information resources for Db2 11 for z/OS and related products

Information about Db2 11 for z/OS and products that you might use in conjunction with Db2 11 is available online in IBM Knowledge Center.

You can find the complete set of product documentation for Db2 11 for z/OS in IBM Knowledge Center.

You can also download other PDF format manuals for Db2 11 for z/OS from IBM Knowledge Center in PDF format manuals for Db2 11 for z/OS (Db2 Home).

© Copyright IBM Corp. 2001, 2020

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on 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 give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

For license inquiries regarding double-byte (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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites 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 which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

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.

© Copyright IBM Corp. 2001, 2020

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.

This information contains 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 actual people or business enterprises 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 shown below:

© (*your company name*) (*year*).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. (*enter the year or years*).

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This information is intended to help you to learn about and plan to use Db2 11 for z/OS. This information also documents General-use Programming Interface and Associated Guidance Information provided by Db2 11 for z/OS.

General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of Db2 11 for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

GUPI General-use Programming Interface and Associated Guidance Information... GUPI

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered marks 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: http://www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

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.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Glossary

The glossary is available in IBM Knowledge Center.

See the Glossary topic for definitions of Db2 for z/OS terms.

© Copyright IBM Corp. 2001, 2020

Index

Numerics

64-bit storage 287

A

access control
    data set protection 185
    multilevel security 188
    outside Db2 185
    within Db2 185
access paths
    defined 220
    determining 225
    using EXPLAIN 221
accessibility
    keyboard xi
    shortcut keys xi
active logs 40
activity sample table 319
address spaces 289
administration
    information roadmap 375
administrative authority 186
aggregate functions
    aggregate values
        calculating 60
    nesting scalar functions 63
aliases 33
ALTDATE user-defined function
    sample 22
ALTER TABLE statement
    check constraints
        defining 29
    DEFAULT clause 27
    referential structures 124
ALTTIME user-defined function
    sample 22
AND operator
    described 69
    using NOT with 69
    using parentheses with 69
application designers
    promoting concurrency 218
application development
    information roadmap 381
application development tools
    Data Studio 233, 308
    Development add-in for Visual Studio .NET 233, 308
    ODBC based workstation tools 233, 308
    Rational Application Developer for WebSphere Software 233, 308
    Rational Developer for System z 233, 308
    WebSphere Studio Application Developer 233, 308
application packages
    privileges 186
application performance
application performance *(continued)*
    analyzing 222
application plans
    privileges 182, 186
application process
    initial state in distributed unit of work 142
    initial state in remote unit of work 144
    state transitions 142
application processes 42, 43, 127
application programming
    performance
        for application programmers 236
        recommendations 236
    performance recommendations 236
application programs
    bind 238
    compile 238
    CURRENT PACKAGE PATH special register 238
    data communication coding 292
    database request module (DBRM) 238
    link-edit 238
    load module 238
    package 238
    precompile 238
    precompiler 238
    preparing 238
    recovery 43, 127
    running 238
    special registers 238
    using as stored procedures 252
    writing 229, 235
application requester
    definition of 140
application server
    definition of 140
application-period temporal tables 89, 95
archive logs 40
archive table
    description 92
archive tables 89
archive-enabled table
    description 92
archive-enabled tables 89
ARCHIVESENSITIVE bind option
    effect 92
AS clause 56
ASCII
    definition 131
    encoding schemes 19
Assembler
    stored procedures
        programming 253
    use with static SQL 241
associative tables 279
attachment facilities 291
attributes
    choosing data types 272

attributes *(continued)*
default values 272
domains
defining 272
naming 272
null values 272
values
determining 272
authentication
CONNECT statement 185
mechanisms 184
authorization hierarchy 186
authorization IDs
CURRENT SQLID 182
explicit privileges 186
primary 182
privileges
administrative authority 182
application plans 182
authorization hierarchy 186
granting 189
objects 182
packages 182
related privileges 182
revoking 189
secondary 182
three-part names 149
automatic query rewrite 94
auxiliary tables 89
availability features 297
AVG function 60

B

backup and recovery
BACKUP SYSTEM utility 193
backups and data checks
scheduling 195
bootstrap data set (BSDS) 193
commit operations 196
coordinators 197
COPY utility 191
data consistency 195
database changes 195
disaster recovery 198
log usage 193
maintaining consistency 197
optimizing availability 198
overview 191
participants 197
RECOVER utility 191, 197
REORG utility 191
REPORT utility 191
rollback operations 196
tools 193
utilities 193
base tables
creating 92
basic information
information roadmap 369
beginners
information roadmap 369
binding
options 151
BIT string subtype 19
bitemporal tables 89, 95
BLOB
length 19
LOB data type 24
block fetch
continuous 154
improving performance 154
limited 154
bootstrap data set (BSDS)
overview 40
usage 193
buffer pools
described 41
size 210
business intelligence 299
business rules
applying to relationships 272
enforcing 18, 34, 125
triggers 34

C

C
stored procedures
programming 253
use with static SQL 241
C++
stored procedures
programming 253
use with static SQL 241
caching data 210
CAF (Call Attachment Facility) 294
Call Level Interface (CLI) 6
CALL statement
execution methods 257
stored procedures
invoking 252, 257
CASE expressions 64
castout process 166
catalog tables
SYSIBM.SYSCOPY 197
SYSIBM.SYSDATABASE 120
SYSIBM.SYSINDEXPART 213
SYSIBM.SYSTABLEPART 213
SYSIBM.SYSTABLES 89
catalogs 38
CCSID (coded character set identifier)
definition 131
Definition 135
description 131
CCSID (Coded Character Set Identifier)
of strings 135
CDB (communications database) 38
character conversion
ASCII 131
character set 131
code page 131
code point 131
coded character set 131
contracting conversion 140
description 131
EBCDIC 131
encoding scheme 131

character conversion *(continued)*
expanding conversion 140
substitution character 131
Unicode 131
UTF-16 131
UTF-8 131
Character conversion
Coded character sets and ccsids 135
character set 131
character strings 19, 68
check constraints
column values
enforcing validity 29
inserting rows into tables 30
updating tables 30
CHECK DATA utility 122, 125, 195
CHECK INDEX utility 195
CHECK utility 195
CICS (Customer Information Control System)
attachment facility 292
operating
recovery from system failure 292
XRF (extended recovery facility) 292
CLI (Call Level Interface) 6
client APIs
Db2 Database Add-Ins for Visual Studio 313
JDBC 313
ODBC 313
SQLJ 313
web services 313
CLOB
LOB data type 24
cluster technology
Db2 product support 302
Parallel Sysplex 156, 287
clustering indexes
implementing 108
performance considerations 213
COALESCE function 83
COBOL
stored procedures
programming 253
use with static SQL 241
code page 131
code point 131
coded character set 131
column definitions
components of
datetime data types 22
ROWID data type 25
string data types 19
distinct types 25
large object (LOB) data types 24
column functions 60
columns
as sort keys 72
calculating values 59
choosing a data type 19
values
enforcing validity 29
commands
how to issue 174
types 174
commit operations
commit operations *(continued)*
points of consistency 196
commit processing 141
COMMIT statement 196
communications database (CDB) 38, 185
comparison operators 68
complex queries 158
composite keys 12
COMPRESS clause
ALTER TABLESPACE statement 212
compression of data 212
concepts 3
concurrency
application design
concurrency recommendations 220
basic recommendations
for application designers 220
for database designers 220
database design
concurrency recommendations 220
locking 215
concurrency control
promoting
application designers 218
database designers 218
CONNECT statement
authenticating users 184
DRDA access 147
explicit 149
connectable and connected state 144
connectable and unconnected state 144
connected state 143
connection
application process states 143, 144
definition of 141
initial state in distributed unit of work 142
management in distributed unit of work 142
management in remote unit of work 144
SQL state
in a distributed unit of work 142
state transitions 142
when ended in a distributed unit of work 143
constraints
types 14
continuous block fetch 154
continuous operation
restart processing 195
coordinating updates
transaction manager support 152
two-phase commit
servers not supporting 153
servers supporting 153
COUNT function 60
coupling facility
coupling facility resource management (CFRM) policies 162
CREATE ALIAS statement 33
CREATE AUXILIARY TABLE statement 119
CREATE DATABASE statement
INDEXBP clause 120
CREATE DISTINCT TYPE statement 25
CREATE FUNCTION statement 64
CREATE GLOBAL TEMPORARY TABLE statement 89
CREATE INDEX statement

CREATE INDEX statement *(continued)*
CLUSTER clause 108
NOT PADDED clause 110
PADDED clause 110
CREATE LOB TABLE statement 119
CREATE PROCEDURE statement 257
CREATE SEQUENCE statement 37
CREATE STOGROUP statement 98
CREATE TABLE statement
base tables
creating 92
BUFFERPOOL clause 120
check constraints
defining 29
DEFAULT clause 27
FOREIGN KEY clause 124
LOB columns, defining 119
NOT NULL clause 26
PARTITION BY clause 94, 112
PARTITION ENDING AT clause 94
PRIMARY KEY clause 124
CREATE TABLESPACE statement
COMPRESS clause 212
LOCKSIZE ANY clause 218
CREATE VIEW statement
WITH CHECK OPTION clause 118
created temporary tables
defining 93
current connection state 142
CURRENT PACKAGE PATH special register 238
CURRENT SQLID 182
cursor stability (CS) 218
cursors
row-positioned 155, 245
scrollable 245
Customer Information Control System (CICS)
attachment facility 292

D

data
accessing
not in a table 85
with host structures 244
with host variable arrays 244
with host variables 242
controlling access 185
joining 77
modifying 85–87
reorganizing
guidelines 213
selecting from columns 56
updating 87
data access
authorization IDs
CURRENT SQLID 182
primary 182
privileges 182
secondary 182
controlling access
with views 189
on demand business, and 316
data access control
data set protection 185
data access control *(continued)*
outside Db2 185
within Db2 185
data caching 210
data checks
scheduling 195
data compression 212
Data Language I (DL/I)
batch features 293
data mining 299
data modeling
diagrams 277
entity-relationship model 268, 277
examples 268
overview 268
recommendations 268
tools 277
Unified Modeling Language (UML) 277
data organization
performance 212
data reorganization
clustering 213
free space 212
I/O activity 213
page gaps 213
REORG utility thresholds 213
unused space 213
data replication 312
data servers
middleware components 310
data set protection 185
data sets
managing 181
data sharing
advantages 157, 162
availability
coupling facility availability 167
duplexing group buffer pools 167
during an outage 167
changed data 166
coupling facility 157
data consistency 162
environment 157, 297
flexible configurations 159
flexible decision support systems 159
flexible operational systems 159
group 156
improved availability 157
managing shared data 159
Parallel Sysplex environment 156
scalable growth
transaction rates 158
tasks affected by 167
data sharing environments
update process 163
data sharing groups 156, 295
data structures
databases 48
hash spaces 53
hierarchy 8
index spaces 51
indexes 31
keys 12
table spaces 50

data structures *(continued)*
tables 89
types 8
views 31
Data Studio 233, 308
data types
BIGINT 21
BLOB 19, 24
built-in 19
CHAR 19, 22
CLOB 24
comparing 19
datetime 22, 272
DBCLOB 24
distinct types 25
GRAPHIC 19
large object (LOB) 24
numeric 21, 272
ROWID 25
SMALLINT 21
string
encoding schemes 19
string subtypes 19
VARCHAR
compared to CHAR 19
VARGRAPHIC
storage limit 24
XML 24
data-partitioned secondary index (DPSI) 112
database connection services (DCS) 185
database descriptors
contents 39
database design
hash access 282
implementing 267
indexes 100, 282
large objects 119
logical
entity-relationship model 268
Unified Modeling Language (UML) 277
logical data modeling 267
physical
entity-relationship model 279
physical data modeling 267
referential constraints 121
table spaces 96
tables 88
database designers
promoting concurrency 218
Database ExplorerQMF for Workstation 7
database request module (DBRM) 45, 130
databases
creating 48, 120
default databases 48
lock operations 48
overview 48
starting 48
stopping 48
users who need their own 48
DataPropagator NonRelational (DPropNR) 293
DATE data type 22
datetime data types 22, 272
Db2 address spaces
ssnmDBM1 289
Db2 address spaces *(continued)*
ssnmDIST 289
ssnmMSTR 289
Db2 Administration Tool for z/OS 308
Db2 Bind Manager 238
Db2 Buffer Pool Analyzer 209
Db2 Connect
Db2 Connect Enterprise Edition 310
Db2 Connect Personal Edition 300, 310
remote servers
accessing 152
requester to remote server 300
DB2 Connect
authentication mechanisms 185
Db2 data servers
DB2 for i 302
Db2 for Linux, UNIX, and Windows 302
Db2 for z/OS 302
Db2 Database
distributed editions 302
Db2 Database Add-Ins for Visual Studio 313
DB2 databases 48
Db2 distributed editions 302
Db2 for i 302
Db2 for Linux, UNIX, and Windows 302
Db2 for z/OS
definition 1
Db2 Interactive (DB2I) 294
Db2 operations
managing 171
Db2 Path Checker 238
Db2 Performance Manager 209
Db2 Query Management Facility (QMF) 48
Db2 Query Management Facility (QMF)QMF for Workstation 7
Db2 Query Monitor 209
Db2 SQL Performance Analyzer 209, 221
Db2 subsystem
local 140
Db2 subsystems
restarting 195, 198
scenarios 297
Db2 Tools for z/OS 305
Db2-defined defaults 27
DB2I (Db2 Interactive)
panels
DB2I Primary Option Menu 173
DBADM authority 88, 186
DBCLOB
LOB data type 24
DBCTRL authority 186
DBD01 directory table space
contents 39
DBMAINT authority 186
DDCS (data definition control support)
database 41
deadlocks
locks 43, 127
uncommitted changes 43, 127
DECFLOAT data type 21
DECIMAL data type 21
DECIMAL function 59, 61
DECLARE CURSOR statement 245
DECLARE GLOBAL TEMPORARY TABLE statement 89, 93

DECLARE statement 93
DECLARE TABLE statement 242
declared temporary tables
  defining 93
default database (DSNDB04)
  defining 48
default values
  compared to null values 28
  Db2-defined defaults 27
  ROWID data type 27
  user-defined default values 27
deferred embedded SQL 5
DELETE statement
  delete rules 123
  role in caching 222
  usage 87
denormalization 279
department sample table 320
dependent rows 14
dependent tables 14
developer
  application programming
    information roadmap 381
Development add-in for Visual Studio .NET 233, 308
development toolsIBM Data Studio 256
DFSMS (Data Facility Storage Management Subsystem) 291
DFSMSdfp
  partitioned data set extended (PDSE) 291
directory
  table space names 39
disability xi
DISABLE ARCHIVE clause
  in ALTER TABLE STATEMENT 92
DISPLAY DATABASE RESTRICT command 213
DISTINCT keyword 56, 60
distinct type 35
distinct types 25
distributed data
  accessing
    communication protocols 184
    communications database (CDB) 147
    Distributed Relational Database Architecture (DRDA) 147
    effects on planning 150
    effects on program preparation 150, 151
    effects on programming 150, 151
    planning considerations 150
    remote servers 148
    resource limit facility 150
  block fetch 154
  connectivity 147
  coordinating updates
    transaction manager support 152
    two-phase commit 153
  network messages
    minimizing 153
  remote servers 146
  rowset fetch 154
distributed data facility (DDF) 295, 300
distributed editions Database
  Db2 302
distributed environments 147
distributed relational database
  definition of 140
Distributed Relational Database Architecture (DRDA)
  connectivity 147
  remote servers
    accessing 147
  security options 184
  web access 300
distributed unit of work
  connection management 142
  definition of 141
domains 272
dormant connection state 142
DOUBLE data type 21
double-byte character set (DBCS) 19
DPropNR (DataPropagator NonRelational) 293
DPSI (data-partitioned secondary index) 112
DSN command of TSO
  command processor
    invoking 294
DSN command processor
  invoking 294
DSN1COMP utility 212
DSNDB04 default database 48
dual logging 40
duplicates
  eliminating 56, 75
  retaining 75
dynamic scrollable cursors 245
dynamic SQL
  applications
    example 247
  description 5
  executing
    with Java 250
    with ODBC functions 248

E

EBCDIC
  definition 131
  encoding schemes 19
employee photo and resume sample table 326
employee sample table 322
employee-to-project activity sample table 329
ENABLE ARCHIVE clause
  in ALTER TABLE STATEMENT 92
encoding scheme
  of strings 135
encoding schemes
  ASCII 19
  EBCDIC 19
  Unicode 19
Enterprise Storage Server (ESS) 287
entities
  defining attributes 272
  defining for relationships 270
  normalizing 274
entity integrity 14
equality
  selecting rows 67
  set of columns
    testing 67
  testing 67
ETL capabilities
  WebSphere DataStage 313

ETL capabilities *(continued)*
WebSphere QualityStage 313
exclusive lock (X-lock) 215
EXECUTE statement 248
exit routines 293
EXPLAIN 221
EXPLAIN statement
access paths
determining 225
EXPLAIN tool 209
explicit privileges 186
expression-based 110
expression-based index 110
expressions
indexes 110
external functions 126
external SQL procedures 36
external stored procedures 36

F

federated database support
defined 304
WebSphere Information Integrator 312
fetch operations
block fetch 154
multiple-row fetch 154, 245
rowset fetches 155
FETCH statement 245
FICON channels 287
field-level access control 189
field-level sensitivity 189
first normal form 274
foreign keys 12
Fortran 235, 241
fourth normal form 274
FREEPAGE clause
data storage 212
index storage 212
FROM clause 58, 77
full outer joins
examples 83
functions
CHAR function 61
MAX function 60
SQLExecDirect() function 248
SQLPrepare() function 248
SUM function 60

G

general-use programming information, described 388
GET DIAGNOSTICS statement 247
GRANT statement 189
GRAPHIC data type 19
graphic strings 19
group buffer pools
described 41
duplexing 163, 167
GBP-dependent 162
GROUP BY clause
queries
simplifying 222
GROUP BY clause *(continued)*
usage 74
GUPI symbols 388

H

hash access 53, 228, 282
hash spaces 53
HAVING clause 58, 76
held connection state 142
high availability 156
history tables 89
host structures
accessing data 244
host variable arrays
accessing data 244
host variables
accessing data 85, 242
role in application performance 222
HTML (Hypertext Markup Language) 258
HTTP (Hypertext Transfer Protocol) 259

I

I/O activity
table spaces 213
IBM Data Studio 307
IBM Z
Linux 302
identity columns 21
IMS (Information Management System)
application programming 293
attachment facility 293
connecting to Db2 293
recovery from system failure 293
system administration 293
IN predicate 71
index attributes 105
index spaces 51
index types
unique indexes 111
indexes
access through 222
attributes
partitioned tables 112
backward index scan 104
clustering 108
compressing 111
creating 100
defining
with composite keys 105
forward index scan 104
nonpartitioned 112
nonunique 108
not padded
advantages 110
disadvantages 110
index-only access 110
varying-length columns 110
NULL keys
excluding 109
padded 110
partitioned 112

indexes *(continued)*
secondary
data-partitioned secondary index (DPSI) 112
nonpartitioned secondary index (NPSI) 112
selecting columns 282
selecting expressions 282
sorts
avoiding 104, 222
unique 106
inequality
selecting rows 67
set of columns
testing 67
testing 67
information roadmap
administration 375
application development 381
Db2 for z/OS 369
new-user 369
programming 381
inner joins 79, 118
insensitive scrollable cursors 245
INSERT privilege 186
INSERT statement
base tables
creating 92
check constraints 30
clustering indexes 108
referencing views 118
role in caching 222
INTEGER data type 21
integrated development environments (IDEs)
Data Studio 233, 308
Development add-in for Visual Studio .NET 233, 308
ODBC based workstation tools 233, 308
Rational Application Developer for WebSphere Software 233, 308
Rational Developer for System z 233, 308
WebSphere Studio Application Developer 233, 308
Intelligent Resource Director (IRD) 287
intent lock 215
interactive SQL 5
interfaces
ISPF 171
JCL 171
SDSF 171
SPUFI 171
TSO/E 171
interim result tables 75
IRLM (internal resource lock manager)
administering 290
IS NULL predicate 26
isolation levels
cursor stability (CS) 218
read stability (RS) 218
repeatable read (RR) 218
uncommitted read (UR) 218
ISPF
user interface 171
ISPF (Interactive System Productivity Facility)
DB2 considerations 294
DB2I Primary Option Menu 173
requirements 293
system administration 294
issuing
commands 174

J

J2EE
WebSphere Application Server 311
Java
dynamic SQL
executing 250
EJB 258
JDK (Java Development Kit) 251
JSP 258
servlets 258
static SQL
executing 250
stored procedures
programming 253
support 313
JCL
user interface 171
JDBC
advantages 251
compared to SQLJ 250, 251
examples 251
middleware 313
static SQL applications
dynamic SQL applications 251
joins
example tables 77
full outer joins 77, 83
inner joins 79
left outer joins 77, 81
overview 77
right outer joins 77, 82

K

Kerberos security 185
keys
composite keys 12
foreign keys 12
parent keys 12
primary keys 12
sort keys 72
unique keys 12

L

large object table spaces 50
large objects (LOBs)
creation of 119
data types 24
leaf pages 213
left outer joins 77, 81
LIKE predicate 68
limited block fetch 154
links
non-IBM Web sites 389
Linux
IBM Z 302
LOAD utility
base tables

LOAD utility *(continued)*
base tables *(continued)*
creating 92
collecting statistics 213
referential constraints
enforcing 122
local area networks (LANs) 303
local Db2 subsystem 140
LOCATION names
three-part names 149
locking
concurrency
promoting 215
deadlocks 215
exclusive lock (X-lock) 215
performance
improving 215
scenarios 215
share lock (S-lock) 215
suspension 215
timeout 215
update lock (U-lock) 215
locks 43, 127
log range directory 39
logical data modeling 267
logical database design
attributes
choosing data types 272
naming 272
values 272
business rules
applying to relationships 272
data modeling
examples 268
recommendations 268
Unified Modeling Language (UML) 277
entities
defining attributes 272
defining for relationships 270
normalizing 274
many-to-many relationships 270
logs
archiving 198
recovery usage 198

M

management tools
Data Studio 305
Db2 family 305
Db2 Tools for z/OS 305
materialized query tables
creating 94
implementing 89
performance 222
merge statements 87
middleware components 313
MIN function 60
mixed data character string columns 19
MIXED string subtype 19
multilevel security 188
multiple-row fetch
defined 154
improving performance 154

N

n-tier architecture 259
native SQL procedures 36
network messages
minimizing 153
new users
information roadmap 369
nonpartitioned secondary index (NPSI) 112
nonunique indexes 108
normal forms 274
normalization
avoiding redundancy 274
first normal form 274
fourth normal form 274
second normal form 274
third normal form 274
NOT keyword
with comparison operators 68
NOT operator
described 69
using with AND and OR 69
NPSI (nonpartitioned secondary index) 112
NULL keys
excluding 109
null values
compared to default values 28
excluding rows 66
retrieving rows 66
usage 26
NULLIF function 61
numeric data types
DECIMAL 21
DOUBLE 21
identity columns 21
INTEGER 21
REAL 21
SMALLINT 21

O

Object Management Group 277
OBJECT names
three-part names 149
object privileges 182
ODBC (Open Database Connectivity)
dynamic SQL
executing 248
examples 248
middleware 313
ODBC based workstation tools 233, 308
OMEGAMON 209
on demand business
described 315
one-phase commit 141
Open Database Connectivity (ODBC) 6
open standards 315
operating environments
Db2 287
mobile 302
z/OS 287
operations tools
Db2 Administration Tool for z/OS 308
OPTIMIZE FOR n ROWS clause

OPTIMIZE FOR n ROWS clause *(continued)*
SELECT statement 155
OR operator
described 69
using NOT with 69
using parentheses with 69
ORDER BY clause
column names
specifying 72
expressions
specifying 74
more than one column
specifying 73
rows
ascending order 73
descending order 73
sorts
avoiding 104
ordering column 72
other information resources
administration 375
application development 381
programming 381
outer joins 118

P

PACKADM authority 186, 189
packages
DRDA access
bind options 151
precompiler options 151
privileges 182
page access 210
page gaps 213
page sets 47
page sizes 283
pages 96, 283
Palm Operating System
Db2 Everyplace 302
parallel processing 222
Parallel Sysplex
benefits 162
coupling facility 156
environment 156, 295
group buffer pool 41
Sysplex Timer 156
parent keys 12, 14
parent rows 14
parent tables 14
partitioned data set extended (PDSE) 291
partitioned data sets
managing 291
partitioned indexes 112
partitioned table spaces
rebalancing data 213
partitioned tables
index attributes 112
partitioning
table-controlled 94
PCTFREE clause 212
performance
application design 208
concurrency control 218
performance *(continued)*
data organization 212
locking 215
managing 207
performance analysis tools 209
performance objectives 207
problem determination 208
problems 207
query performance
access paths 220
analyzing 222
described 220
EXPLAIN 221
performance objectives
requirements 207
performance tools
Db2 Buffer Pool Analyzer 209
Db2 Performance Expert 209
Db2 Query Monitor 209
Db2 SQL Performance Analyzer 209
performance toolsDb2 SQL Performance Analyzer for z/OS 221
physical data modeling 267
physical database design
associative tables 279
customizing views 281
denormalization of tables 279
determining what columns to index 282
determining what expressions to index 282
PL/I
stored procedures
programming 253
use with static SQL 241
plan tables 225
points of consistency 43, 128
precompiler options 151
predicates 65
prefetch
sequential 210
PREPARE statement 222
primary authorization IDs 182
primary keys 12
privileges
administrative authority 182
application plans 182
authorization hierarchy 186
explicit 186
granting 189
held by authorization IDs 182
object 182
packages 182
related 182
revoking 189
roles 182
security labels 182
procedures
external SQL procedures 36
external stored procedures 36
native SQL procedures 36
programming
information roadmap 381
programming interface information, described 388
programming languages 235
project activity sample table 328

project sample table 327
pureXML 168, 263

Q

QMF for Workstation
  Database Explorer feature 7
  query results 7
  query-related features 7
  SQL statements
    entering and processing 7
queries
  coding 222
  for which table reference does not matter 339
query performance
  access paths 220
  accessing remote servers 154
  analyzing 222
  block fetch 154
  described 220
  FETCH FIRST n ROWS ONLY 155
  OPTIMIZE FOR n ROWS 155
  result sets
    optimizing 155
  rowset-positioned cursors 155
query performanceDb2 SQL Performance Analyzer for z/OS 221

R

Rational Application Developer for WebSphere Software 233, 308
Rational Data Architect 277
Rational Developer for System z 233, 308
Rational Rapid Developer 277
Rational Rose Data Modeler 277
read stability (RS) 218
REAL data type 21
REBUILD INDEX utility 213
record identifiers (RIDs) 106, 213
record lengths
  page sizes 283
records 283
RECOVER utility
  recovering page sets 197
recovery
  restoring data consistency 43, 127
  See backup and recovery 197
  unit of 44, 128
recovery planning
  continuous operation 156
referential constraints
  delete rules 123
  enforcing 122
  exception tables 125
  implementing 121
  insert rules 122
  loading tables 125
  referential structures
    building 124
  update rules 123
referential integrity
  delete rules 123
referential integrity *(continued)*
  enforcing 122
  exception tables 125
  implementing 121
  insert rules 122
  loading tables 125
  referential structures
    building 124
  update rules 123
referential structures
  building 124
related privileges 182
release-pending connection state 142
remote database server
  definition of 140
remote servers
  accessing
    explicit CONNECT statements 149
    three-part names 149
    with aliases 33
  Db2 Connect 152
  described 146
  DRDA access 147
  query efficiency 154
remote unit of work
  connection management 144
  definition of 144
REORG utility
  collecting statistics 213
  reorganizing data
    guidelines 213
  thresholds 213
REORG-pending status 213
repeatable read (RR) 218
replication 312
REPORT utility 193
Representational State Transfer (REST) 316
requesters 146
Resource Access Control Facility (RACF) 290
resource limit facility (governor)
  database 41
restart 198
restart processing 195
result columns
  naming 56
result sets
  optimizing 155
result tables
  interim 75
REUSASID=YES keyword 289
REVOKE statement 189
REXX
  stored procedures
    programming 253
  use with static SQL 241
right outer joins 77, 82
rollback operations
  savepoints 196
ROLLBACK statement 196
routines
  types 35
ROWID data type
  default values 27
rows

rows *(continued)*
deleting 88
description 3
designing
wasted space 283
inserting
with check constraints 30
ordering 72
page sizes 283
record lengths 283
results
ascending order 73
descending order 73
rowsets 155, 245
RRS (Resource Recovery Services) 294
RUNSTATS utility
collecting statistics 213
system tuning 195

S

sample applications
databases 337
storage 336
storage groups 337
structure 336
sample tables
DSN8B10.ACT (activity) 319
DSN8B10.DEMO_UNICODE (Unicode sample) 330
DSN8B10.DEPT (department) 320
DSN8B10.EMP (employee) 322
DSN8B10.EMP_PHOTO_RESUME (employee photo and resume) 326
DSN8B10.EMPPROJACT (employee-to-project activity) 329
DSN8B10.PROJ (project) 327
PROJACT (project activity) 328
relationships 331
storage 336
views 332
savepoints 196
SBCS string subtype 19
scalability features 297
scalable growth
with data sharing 158
without data sharing 158
scalar functions
CHAR 61
DECIMAL 61
nesting aggregate functions 63
NULLIF 61
user-defined 126
YEAR 61
schema names 10
schema qualifiers 10
schemas 10, 267
scrollable cursors
dynamic 245
insensitive 245
sensitive 245
SCT02 table space 39
SDSF
user interface 171
search conditions
search conditions *(continued)*
specifying 76
second normal form 274
secondary authorization IDs 182
secondary group buffer pools 163
security
authentication 184
authentication mechanisms 184
authorization IDs 181
communications database (CDB) 185
controlling access
with views 189
Db2 subsystems
accessing 184
field-level access control 189
field-level sensitivity 189
Kerberos 185
local access
security checks 184
multilevel security 188
privileges 181
RACF 184
remote access
security checks 184
Resource Access Control Facility (RACF) 290
z/OS Security Server 184, 287, 290
segmented table spaces 50
SELECT privilege 186
SELECT statement
BETWEEN predicate 71
OPTIMIZE FOR n ROWS clause 155
processing 58, 117
role in caching 222
selecting data from columns
SELECT clause
SELECT * 56
SELECT column-name 56
SELECT expression 56
self-referencing tables 14
sensitive scrollable cursors 245
sequences 37
sequential prefetch 210
server
remote 140
server-side programming
described 258
using WebSphere Application Server 258
servers
Db2 for z/OS
benefits 261
local access 184
remote access
with DRDA 147
workstation access 185
service-oriented architecture (SOA) 316
servlets 258
SGML (Standard Generalized Markup Language) 284
share lock (S-lock) 215
shared-nothing architecture 158
shortcut keys
keyboard xi
Simple Object Access Protocol (SOAP) 316
simple table spaces 50
single logging 40

single-byte character set (SBCS) 19
SKCT (skeleton cursor table) 39
skeleton cursor table (SKCT) 39
skeleton package table (SKPT) 39
SKPT (skeleton package table) 39
Smalltalk programming language 235
solid-state drives 287
sort keys 72
sorts
    avoiding 104
sourced functions 126
SPT01 table space 39
SPUFI
    user interface 171
SQL
    applications
        dynamic SQL statements 247
    dynamic statements 247
SQL (structured query language)
    executing 3
SQL (Structured Query Language)
    Call Level Interface (CLI) 6
    deferred embedded 5
    dynamic 5
    JDBC 7
    Open Database Connectivity (ODBC) 6
    SQLJ 7
    static 4
SQL communication area (SQLCA) 247
SQL functions 126
SQL procedural language
    example 256
SQL statements
    execution
        checking 247
    modifying data 85
SQL/OLB 7
SQLCA (SQL communication area) 247
SQLExecute() function 248
SQLJ
    compared to JDBC 250, 251
    dynamic SQL applications 251
    examples 251
    middleware 313
    static SQL applications 251
SQLPrepare() function 248
ssnmDBM1 address space 289
ssnmDIST address space 289
ssnmMSTR address space 289
state
    application process 143, 144
    SQL connection 142
statement tables 225
static SQL
    applications
        overview 241
        writing 241
    checking execution 247
    DECLARE TABLE statement 242
    executing
        with Java 250
    host structures
        accessing data 244
    host variable arrays
static SQL *(continued)*
    host variable arrays *(continued)*
        accessing data 244
    host variables
        accessing data 242
    rows
        retrieving 245
    table definitions 242
    view definitions 242
storage
    64-bit 287
    assigning table spaces 98
    Intelligent Resource Director (IRD) 287
    SMS-managed 98
    Storage Management Subsystem (SMS) 98
storage groups
    for sample applications 337
Storage Management Subsystem (SMS) 98, 291
storage structures
    index spaces 47
    table spaces 47
stored procedures
    authorization requirements 257
    calling 257
    CREATE PROCEDURE statement 256
    creating
        programming languages 253
        with development tools 256
    developing
        with tools 256
    environment 257
    external SQL procedures 36
    external stored procedures 36
    FETCH statement 256
    native SQL procedures 36
    preparing 257
    processing with 253
    processing without 253
    using application programs 252
    writing
        SQL procedural language 256
string
    CCSID 135
    conversion 131
    description 131
    encoding scheme 135
string data types
    encoding schemes 19
string datatypes 272
strings
    concatenating 59
structured query language (SQL)
    AND operator 69
    executing
        from a workstation 5
        interactive SQL 5
    GROUP BY clause 74, 76
    HAVING clause 76
    joins
        full outer joins 77, 83
        inner joins 79
        left outer joins 77, 81
        right outer joins 77, 82
    modifying data

structured query language (SQL) *(continued)*
modifying data *(continued)*
DELETE statement 87
INSERT statement 86
NOT operator
described 69
using with AND and OR 69
OR operator
described 69
using NOT with 69
using parentheses with 69
ORDER BY clause
expressions 74
more than one column 73
specifying column names 72
procedural language 253
QMF for Workstation 7
UNION keyword
eliminating duplicates 75
retaining duplicates 75
UPDATE statement
modifying data 87
using NOT with 69
using parentheses with 69
WHERE clause
aggregate functions 60
filtering rows 65
processing order in SELECT statement 58
writing queries
aggregate functions 60
AS clause 56
calculating aggregate values 60
calculating values in columns 59
CASE expressions 64
column functions 60
CONCAT keyword 59
DECIMAL function 59, 61
DISTINCT keyword 56, 60
eliminating duplicate rows 56
functions and expressions 59
naming result columns 56
predicates 65
processing SELECT statements 58
scalar functions 61
search condition 65
selecting data from columns 56
subqueries 58, 84
subselects 58
user-defined functions 64
structures
hierarchy 98
subqueries 58, 84
subselects 58
substitution character 131
subsystems
Db2 289
z/OS 289
SYSADM authority 186, 189
SYSCTRL authority 186, 189
SYSDUMMY1 339
SYSDUMMYA 339
SYSDUMMYE 339
SYSDUMMYU 339
SYSIBM.SYSCOPY catalog table 197
SYSIBM.SYSDATABASE catalog table 120
SYSIBM.SYSINDEXPART catalog table 213
SYSIBM.SYSSTOGROUP catalog table 98
SYSIBM.SYSTABLEPART catalog table 213
SYSIBM.SYSTABLES catalog table 89
SYSIBM.SYSVOLUMES catalog table 98
SYSIBMADM.GET_ARCHIVE global variable
effect 92
SYSLGRNX directory table
table space 39
SYSOPR authority 186, 189
Sysplex Timer 156
system objects 38
system resources
tuning 210
system schemas 10
system structures
active logs 40
archive logs 40
bootstrap data set (BSDS) 40
buffer pools 41
catalog tables 38
catalogs 38
system-period data versioning 95
system-period temporal tables 89, 95
Systems Network Architecture (SNA) 295, 303
SYSUTILX directory table space 39

T

table functions
user-defined 126
table spaces
assigning to physical storage 98
creating 96
for sample applications 337
large object 50
partitioned 50
scanning 222
segmented 50
simple 50
universal 50
XML 50
table-controlled partitioning 94
tables
application-period temporal 89
archive 89
archive-enabled 89
associative 279
auxiliary 89
base 89
bitemporal 89
column definitions
choosing a data type 19
components 18
created temporary tables 93
creating 88
declared temporary tables 93
denormalization 279
dependent 14
example tables
DEPT 55
EMP 55
EMPPROJACT 55

tables *(continued)*
example tables *(continued)*
PARTS 77
PRODUCTS 77
PROJ 55
exception tables 125
history 89
inserting rows
check constraints 30
joining 118
materialized query 89
overview 11
plan tables 225
referential structures 124
result 89
rows
deleting 88
self-referencing 14
statement tables 225
system-period data versioning 95
system-period temporal 89
temporal 89, 95
temporary 89
types 89
updating with check constraints 30
TCP/IP 303
temporal tables
creating 95
temporary tables
creating 93
third normal form 274
threads 253, 295
three-part names 149
TIME data type 22
Time Sharing Option (TSO)
attachment facility 292
TIMESTAMP data type 22
tools
backup and recovery 193
Data Server Manager 305
Db2 Administration Tool for z/OS 308
DB2 Bind Manager 238
Db2 Path Checker 238
toolsIBM Data Studio 307
transactions 42
triggers
creating 125
overview 34
TRUNCATE statement 88
truncate statements 88
trusted context 182
TSO (Time Sharing Option)
application programs
batch 294
foreground 294
attachment facility 292, 294
Db2 considerations 294
requirements 293
TSO/E
user interface 171
tutorial
utilities
running from DB2I 354, 361
writing JCL job 364
two-phase commit
servers not supporting 153
servers supporting 153
two-tier architecture 259

U

uncommitted read (UR) 218
unconnectable and connected state 144
unconnectable and unconnected state 144
unconnected state 143
Unicode
definition 131
encoding schemes 19
sample table 330
Unified Modeling Language (UML)
tools
Rational Data Architect 277
Rational Rapid Developer 277
Rational Rose Data Modeler 277
WebSphere Business Integration Workbench 277
WebSphere Studio Application Developer 277
UNION keyword
duplicates
eliminating 75
retaining 75
unique constraints 14
unique indexes
implementing 106
unique keys 12
unit of recovery 43, 44, 127, 128
unit of work
ending 43, 128
initiating 43, 128
Universal Description, Discovery, and Integration (UDDI) 316
universal table spaces (UTS) 50
update lock (U-lock) 215
UPDATE privilege 186
UPDATE statement
modifying data 87
role in caching 222
updates and check constraints 30
updates
coordinating 152
user-defined default values 27
user-defined functions
creating 126
external 126
overview 64
samples
ALTDATE 22
ALTTIME 22
sourced 126
SQL 126
writing queries 64
user-defined scalar functions 126
user-defined table functions 126
user-defined types 35
UTF-16 131
UTF-8 131
utilities
running from DB2I
tutorial 354, 361
tutorial 353

utilities *(continued)*
writing JCL job
tutorial 364

V

varying-length columns 68
view definitions
single table 117
views
accessing data 189
creating 117
customizing 281
inserting data 118
joining data 118
overview 31
updating data 118

W

web applications
developing 233, 261, 308
web services
XML 316
Web Services Description Language (WSDL) 316
WebSphere Application Server 258, 311
WebSphere Business Integration Workbench 277
WebSphere DataStage
ETL capabilities 313
WebSphere Host Integration 312
WebSphere Information Integrator 304, 312
WebSphere MQ 152
WebSphere products 310
WebSphere QualityStage
ETL capabilities 313
WebSphere Studio Application Developer
developing web applications 233, 261, 308
UML data modeling 277
WebSphere Studio product family 311
WHERE clause
aggregate functions 60
filtering rows 65
processing order in SELECT statement 58
WITH HOLD 222
work file database
description 42
Workload Manager (WLM) 158, 261
World Wide Web 315

X

XML (Extensible Markup Language)
Db2 for z/OS support 263
DB2 for z/OS support 168
pureXML 168, 263
SQL/XML functions 285
usage 284
web services support 284, 285
XML support 285
XML table spaces 50

Z

z/OS Security Server 290



Product Number: 5615-DB2
5697-P43

SC19-4058-10

