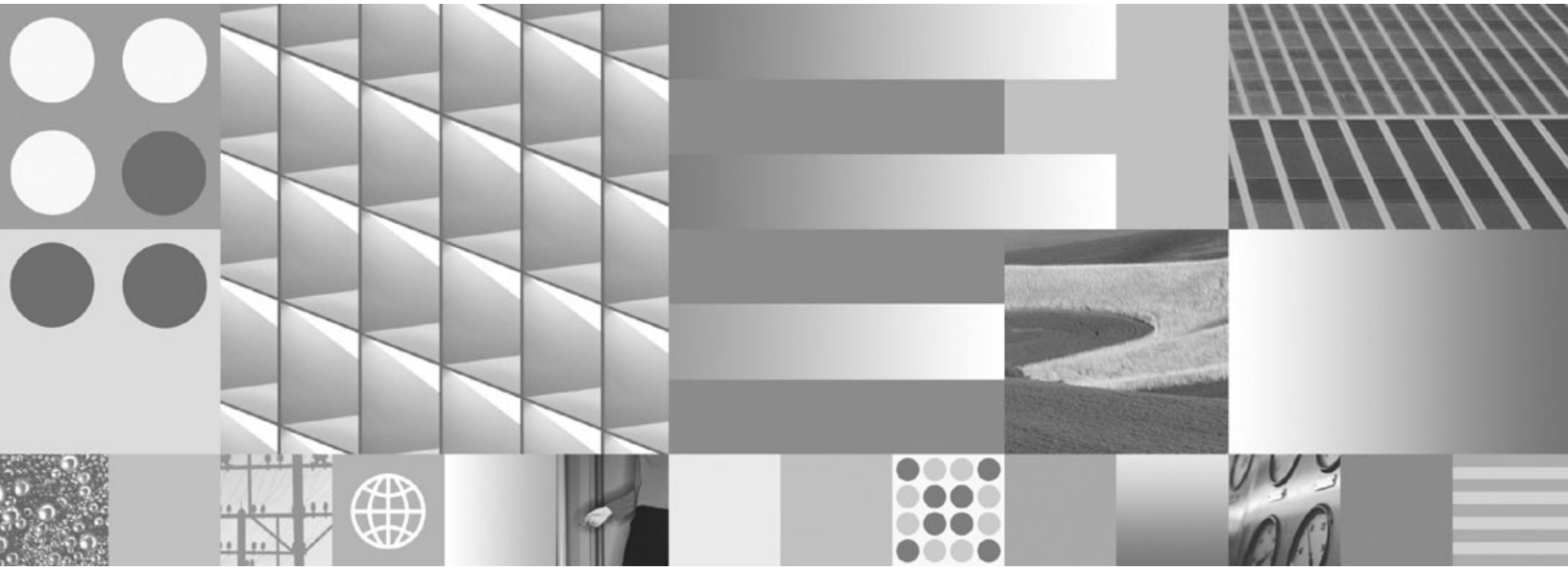
DB2 Version 9.5
for Linux, UNIX, and Windows

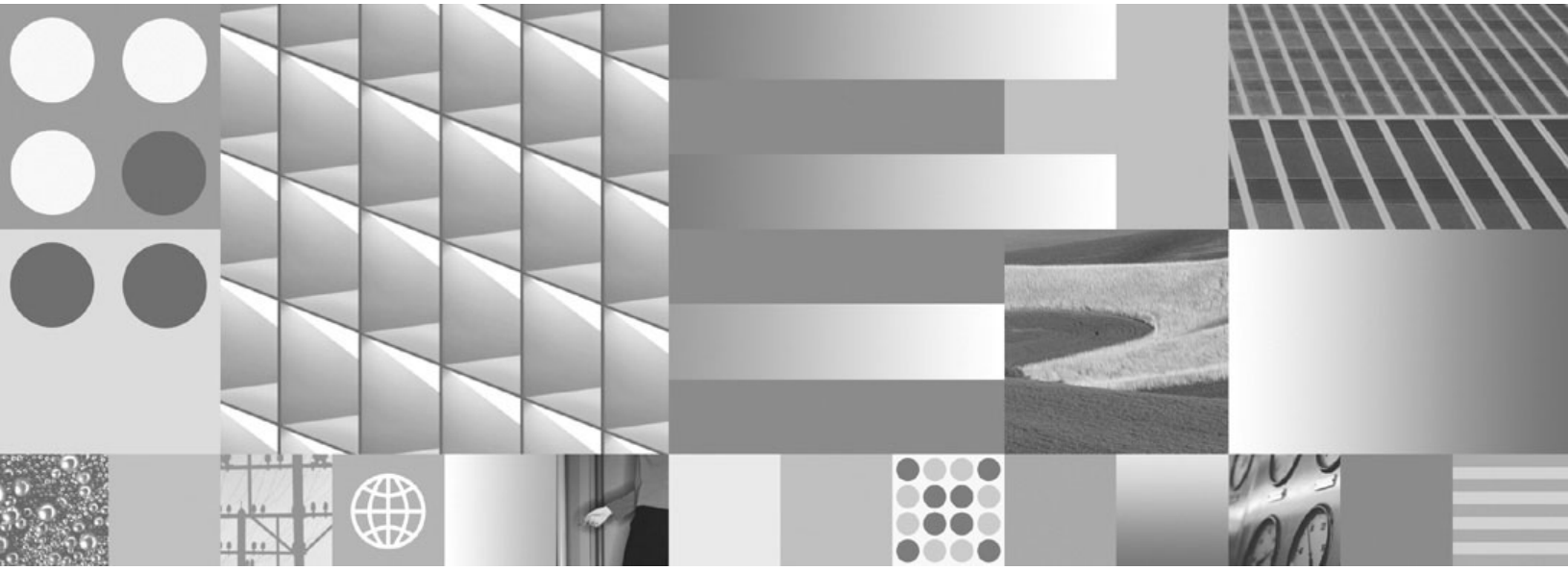


Data Movement Utilities Guide and Reference

Updated March, 2008

SC23-5847-01

DB2 Version 9.5
for Linux, UNIX, and Windows



Data Movement Utilities Guide and Reference

Updated March, 2008

SC23-5847-01

Note
Before using this information and the product it supports, read the general information under Appendix F, "Notices," on page 453.

Edition Notice

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

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

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

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

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

© Copyright International Business Machines Corporation 1993, 2008. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book v

Part 1. Data movement utilities and reference 1

Chapter 1. Data movement options available in DB2 V9.5 3

Chapter 2. Export utility 7
Export utility overview 7
Privileges and authorities required to use the export utility 8
Exporting data 8
Exporting XML data 9
LBAC-protected data export considerations . . . 12
Table export considerations 13
Typed table export considerations 14
Identity column export considerations 16
LOB export considerations 16
Reference - Export 18
EXPORT 18
EXPORT command using the ADMIN_CMD procedure 27
db2Export - Export data from a database . . . 36
Export sessions - CLP examples 42

Chapter 3. Import utility 45
Import overview 45
Privileges and authorities required to use import . . 47
Importing data 48
Importing XML data 50
Imported table re-creation 50
Typed table import considerations 52
LBAC-protected data import considerations . . 55
Buffered-insert imports 57
Identity column import considerations 58
Generated column import considerations . . . 59
LOB import considerations 60
User-defined distinct types import considerations 61
Additional considerations for import 61
Client/server environments and import 61
Table locking during import 62
Reference - Import 63
IMPORT 63
IMPORT command using the ADMIN_CMD procedure 86
db2Import - Import data into a table, hierarchy, nickname or view 109
Import sessions - CLP examples 122

Chapter 4. Load utility 125
Load overview 125
Privileges and authorities required to use load . . 128
LOAD authority 129
Loading data 129
Loading XML data 131
Load considerations for partitioned tables . . . 132
LBAC-protected data load considerations . . . 134
Identity column load considerations 136
Generated column load considerations 138
Moving data using the CURSOR file type . . . 141
Propagating dependent immediate staging tables 143
Refreshing dependent immediate materialized query tables 144
Multidimensional clustering considerations . . 145
Moving data using a customized application (user exit) 146
Additional considerations for load 152
Parallelism and loading 152
Index creation during load operations 153
Compression dictionary creation during load operations 156
Options for improving load performance . . . 157
Load features for maintaining referential integrity 161
Checking for integrity violations following a load operation 162
Checking for constraint violations using SET INTEGRITY 164
Table locking during load operations 167
Read access load operations 168
Table space states during and after load operations 170
Table states during and after load operations 171
Load exception tables 173
Failed or incomplete loads 174
Restarting an interrupted load operation . . . 174
Restarting or terminating an ALLOW READ ACCESS load operation 175
Recovering data with the load copy location file 176
Load dump file 178
Load temporary files 178
Load utility log records 179
Load overview–partitioned database environments 179
Loading data in a partitioned database environment 181
Monitoring a load operation in a partitioned database environment using the LOAD QUERY command 188
Resuming, restarting, or terminating load operations in a partitioned database environment 189
Migration and version compatibility 191
Reference - Load in a partitioned environment 192
Reference - Load 199
LOAD 199
LOAD command using the ADMIN_CMD procedure 230
db2Load - Load data into a table 261

© Copyright IBM Corp. 1993, 2008

Load sessions - CLP examples 282
SET INTEGRITY 285
LOAD QUERY 302
LIST TABLESPACES 308

Chapter 5. Other data movement options 323
Moving data with DB2 Connect 323
The IBM Replication Tools by Component 325
Copying schemas 326
Examples of schema copy using the db2move utility 327
db2move - Database movement tool 329
Performing a redirected restore using an automatically generated script 337
RESTORE DATABASE 338
High availability through suspended I/O and online split mirror support 355
db2inidb - Initialize a mirrored database . . . 356
db2relocatedb - Relocate database 358
db2look - DB2 statistics and DDL extraction tool 362

Chapter 6. File formats and data types 373
Export/Import/Load utility file formats 373
Moving data across platforms - file format considerations 373
Delimited ASCII (DEL) file format 375
Non-delimited ASCII (ASC) file format 381
PC Version of IXF file format 385
Worksheet File Format (WSF) 422
Unicode considerations for data movement . . . 423
Character set and national language support . . . 425
XML data movement 425
Important considerations for XML data movement 426
LOB and XML file behavior when importing and exporting 427
XML data specifier 428
Query and XPath Data Model 429

Part 2. Appendixes 431

Appendix A. Differences between the import and load utility 433

Appendix B. Bind files used by the export, import, and load utilities . . . 435

Appendix C. How to read the syntax diagrams 437

Appendix D. Collecting data for data movement problems 441

Appendix E. Overview of the DB2 technical information 443
DB2 technical library in hardcopy or PDF format 443
Ordering printed DB2 books 446
Displaying SQL state help from the command line processor 446
Accessing different versions of the DB2 Information Center 447
Displaying topics in your preferred language in the DB2 Information Center 447
Updating the DB2 Information Center installed on your computer or intranet server 448
DB2 tutorials 449
DB2 troubleshooting information 450
Terms and Conditions 450

Appendix F. Notices 453

Index 457

About this book

This book provides information about and shows you how to use the following DB2® Database for Linux®, UNIX®, and Windows® data movement utilities:

- Export and Import

  The export and import utilities move data between a table or view and another database or spreadsheet program; between DB2 databases; and between DB2 databases and host databases using DB2 Connect™. The export utility moves data from a database into operating system files; you can then use those files to import or load that data into another database.

- Load

  The load utility moves data into tables, extends existing indexes, and generates statistics. The load utility moves data much faster than the import utility when large amounts of data are involved. Data exported using the export utility can be loaded using the load utility.

  When the load utility is used in a partitioned database environment, large amounts of data can be distributed and loaded into different database partitions.

For a complete listing of data movement options, see Data movement options available in DB2 V9.5

© Copyright IBM Corp. 1993, 2008

Part 1. Data movement utilities and reference

© Copyright IBM Corp. 1993, 2008

Chapter 1. Data movement options available in DB2 V9.5

There are various data movement options available in DB2 V9.5. The following table provides an overview of the data movement tools and utilities available to you. Use this table as a guide to help you determine which data movement options might best suit your needs.

Table 1. Data movement options available in DB2 V9.5

Utility name	**Load utility**
Purpose	To efficiently move large quantities of data into newly created tables, or into tables that already contain data.
Cross platform compatible	Yes
Best practice usage	This utility is best suited to situations where performance is your primary concern. This utility can be used as an alternative to the import utility. It is faster then the import utility because it writes formatted pages directly into the database rather than using SQL INSERTS. In addition, the load utility allows you the option to not log the data or use the COPY option to save a copy of the loaded data. Load operations can fully exploit resources, such as CPUs and memory on SMP and MPP environments.
References	Loading data

Utility name	**db2move**
Purpose	Using the db2move utility with the COPY option, allows you to copy schema templates (with or without data) from a source database to a target database or move an entire schema from a source database to a target database. Using the db2move utility with the IMPORT or EXPORT option facilitates the movement of a large numbers of tables between DB2 databases.
Cross platform compatible	Yes
Best practice usage	When used with the COPY option, the source and the target database must be different. The COPY option is useful in making schema templates. Use the IMPORT or EXPORT option for cloning databases when there is no support for cross-platform backup and restore operations. The IMPORT and EXPORT options are used in conjunction with the db2look command.
References	• "Copying a schema" in *Data Servers, Databases, and Database Objects Guide* • Imported table re-creation

© IBM Corporation 1993, 2008

Utility name	Import utility
Purpose	To insert data from an external file into a table, hierarchy, view, or nickname
Cross platform compatible	Yes
Best practice usage	The import utility can be a good alternative to the load utility in the following situations: • where the target table is a view • the target table has constraints and you don't want the target table to be put in the Set Integrity Pending state • the target table has triggers and you want them fired
References	Importing data

Utility name	Export utility
Purpose	To export data from a database to one of several external file formats. The data can then be imported or loaded at a later time.
Cross platform compatible	Yes
Best practice usage	This utility is best suited in situations where you want to store data in an external file, to either process it further or move data to another table. High Performance Unload (HPU) is an alternative, however, it must be purchased separately. Export supports XML columns.
References	Exporting data

Utility name	ADMIN_COPY_SCHEMA procedure
Purpose	Allows you to make a copy of all the objects in a single schema and re-create those objects in a new schema. This copy operation can be performed with or without data, within a database.
Cross platform compatible	Yes

Utility name	ADMIN_COPY_SCHEMA procedure
Best practice usage	This utility is useful for making schema templates. It is also useful if you want to experiment with a schema (for example, try out new indexes) without impacting the source schema's behavior. The key differences between the ADMIN_COPY_SCHEMA procedure and the db2move utility are: • The ADMIN_COPY_SCHEMA procedure is used on a single database while the db2move utility is used across databases • The db2move utility fails when invoked if it cannot create a physical object such as a table or index. The ADMIN_COPY_SCHEMA procedure logs errors and continues. • The ADMIN_COPY_SCHEMA procedure uses load from cursor to move data from one schema to the other. The db2move utility uses a remote load, similar to a load from cursor, which pulls in the data from the source database.
References	"Copying a schema" in *Data Servers, Databases, and Database Objects Guide*

Utility name	Restore utility with the REDIRECT option and the GENERATE SCRIPT option
Purpose	To copy an entire database from one system to another using a script from an existing backup image.
Cross platform compatible	Limited. See References
Best practice usage	This utility is best suited in situations where a backup image exists.
References	• "Performing a redirected restore using an automatically generated script" in *Data Recovery and High Availability Guide and Reference* • "Backup and restore operations between different operating systems and hardware platforms" in *Data Recovery and High Availability Guide and Reference*

Utility name	db2relocatedb - Relocate database command
Purpose	To rename a database, or relocate a database or part of a database to the same system or a different system.
Cross platform compatible	No

Utility name	db2relocatedb - Relocate database command
Best practice usage	• This utility can be used for situations where a backup and restore could be time consuming. • This utility is an alternative to using backup and restore to move or create copies of databases. • It also provides a quick method of cloning a database for alternative environments such as testing. • It can be used to move table space containers to a new set of storage devices
References	"db2relocatedb - Relocate database command" in *Command Reference*

Utility name	Split mirror
Purpose	To create a clone, standby, or backup database.
Cross platform compatible	No
Best practice usage	• create a standby system in case of a primary failure to reduce down time • move backup operations away from a live production machine onto a split database • provides a quick method of cloning a database for alternate environments, such as testing
Considerations	• only DMS table spaces can be backed up on the split version of the database • usually used in conjunction with some flashcopy technology provided with storage systems • an alternative is to issue a file copy once the database is suspended, however this duplicates the amount of storage for the database
References	"High availability through online split mirror and suspended I/O support" in *Data Recovery and High Availability Guide and Reference*

Chapter 2. Export utility

Export utility overview

The export utility extracts data using an SQL select or an XQuery statement, and places that information into a file. You can use the output file to move data for a future import or load operation or to make the data accessible for analysis.

The export utility is a relatively simple, yet flexible data movement utility. You can activate it through the Control Center, by issuing the EXPORT command in the CLP, by calling the ADMIN_CMD stored procedure, or by calling the db2Export API through a user application.

The following items are mandatory for a basic export operation:

- The path and name of the operating system file in which you want to store the exported data
- The format of the data in the input file
  Export supports IXF, WSF, and DEL data formats for the output files.
- A specification of the data that is to be exported
  For the majority of export operations, you need to provide a SELECT statement that specifies the data to be retrieved for export. When exporting typed tables, you don't need to issue the SELECT statement explicitly; you only need to specify the subtable traverse order within the hierarchy

You can use the export utility with DB2 Connect if you need to move data in IXF format.

Additional options
There are a number of parameters that allow you to customize an export operation. File type modifiers offer many options such as allowing you to change the format of the data, date and time stamps, or code page, or have certain data types written to separate files. Using the **METHOD** parameters, you can specify different column names to be used for the exported data.

You can export from tables that include one or more columns with an XML data type. Use the **XMLFILE**, **XML TO**, and **XMLSAVESCHEMA** parameters to specify details about how those exported documents are stored.

There are a few ways to improve the export utility's performance. As the export utility is an embedded SQL application and does SQL fetches internally, optimizations that apply to SQL operations apply to the export utility as well. Consider taking advantage of large buffer pools, indexing, and sort heaps. In addition, try to minimize device contention on the output files by placing them away from the containers and log devices.

The messages file
The export utility writes error, warning, and informational messages to standard ASCII text message files. For all interfaces except the CLP, you must specify the name of these files in advance with the **MESSAGES** parameter. If you are using the CLP and do not specify a messages file, the export utility writes the messages to standard output.

© Copyright IBM Corp. 1993, 2008

Privileges and authorities required to use the export utility

Privileges enable you to create, update, delete, or access database resources. Authority levels provide a method of mapping privileges to higher-level database manager maintenance and utility operations.

Together, privileges and authorities control access to the database manager and its database objects. You can access only those objects for which you have the appropriate authorization: that is, the required privilege or authority.

You must have SYSADM or DBADM authority or the CONTROL or SELECT privilege for each table or view participating in the export operation.

When you are exporting LBAC-protected data, the session authorization ID must be allowed to read the rows or columns that you are trying to export. Protected rows that the session authorization ID is not authorized to read are not exported. If the SELECT statement includes any protected columns that the session authorization ID is not allowed to read, the export utility fails, and an error (SQLSTATE 42512) is returned.

Exporting data

Use the export utility to export data from a database to a file. The file can have one of several external file formats. You can specify the data to be exported by supplying an SQL SELECT statement or by providing hierarchical information for typed tables.

You need SYSADM authority, DBADM authority, the CONTROL privilege, or the SELECT privilege on each participating table or view to export data from a database

Before running the export utility, you must be connected (or be able to implicitly connect) to the database from which you will export the data. If implicit connect is enabled, a connection to the default database is established. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be through a direct connection through the engine and not through a DB2 Connect gateway or loop back environment.

Because the utility issues a COMMIT statement, you should complete all transactions and release all locks by issuing a COMMIT or a ROLLBACK statement before running the export utility. There is no requirement for applications accessing the table and using separate connections to disconnect.

The following restriction applies to the export utility:

- This utility does not support tables with structured type columns.

You can run the export utility by using the Export Table notebook in the Control Center or the application programming interface (API) db2Export, or by specifying the EXPORT command in the command line processor (CLP).

Using the Export Table notebook

To export data by using the Export Table notebook:

1. From the Control Center, expand the object tree until you find the Tables or Views folder.

2. Click on the folder that you want to work with. Any existing tables or views are displayed in the pane on the right side of the window (the contents pane).
3. In the contents pane, right-click the table or view that you want, and select Export from the pop-up menu. The Export Table notebook opens.

Detailed information about the Export Table notebook is provided in the Control Center online help facility.

Issuing an EXPORT command by using the CLP

A very simple export operation requires you to specify only a target file, a file format, and a source file for the SELECT statement.

To export data from the CLP, enter the EXPORT command:

```
db2 export to filename of ixf select * from table
```

where `filename` is the name of the output file that you want to create and export, `ixf` is the file format, and `table` is the name of the table that contains the data you want to copy.

However, you might also want to specify a messages file to which warning and error messages will be written. To do that, add the **MESSAGES** parameter and a message file name (in this case, `msg.txt`) so the command is:

```
db2 export to filename of ixf messages msgs.txt select * from table
```

For complete syntax and usage information, see ″EXPORT command.″

Exporting XML data

When exporting XML data, the resulting QDM (XQuery Data Model) instances are written to a file or files separate from the main data file containing exported relational data. This is true even if neither the XMLFILE nor the XML TO option is specified. By default, exported QDM instances are all concatenated to the same XML file. You can use the `XMLINSEPFILES` file type modifier to specify that each QDM instance be written to a separate file.

The XML data, however, is represented in the main data file with an XML data specifier (XDS). The XDS is a string represented as an XML tag named ″XDS″, which has attributes that describe information about the actual XML data in the column; such information includes the name of the file that contains the actual XML data, and the offset and length of the XML data within that file.

The destination paths and base names of the exported XML files can be specified with the XML TO and XMLFILE options. If the XML TO or XMLFILE option is specified, the format of the exported XML file names, stored in the FIL attribute of the XDS, is `xmlfilespec.xxx.xml`, where `xmlfilespec` is the value specified for the XMLFILE option, and `xxx` is a sequence number for xml files produced by the export utility. Otherwise, the format of the exported XML file names is: `exportfilename.xxx.xml`, where `exportfilename` is the name of the exported output file specified for the EXPORT command, and `xxx` is a sequence number for xml files produced by the export utility.

By default, exported XML files are written to the path of the exported data file. The default base name for exported XML files is the name of the exported data file, with an appending 3-digit sequence number, and the `.xml` extension.

Examples

For the following examples, imagine a table USER.T1 containing four columns and two rows:

```
C1 INTEGER
C2 XML
C3 VARCHAR(10)
C4 XML
```

Table 2. USER.T1

C1	C2	C3	C4
2	<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to><from> Me</from><heading>note1</heading><body>Hello World!</body></note>	'char1'	<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</to><from> Her</from><heading>note2</heading><body>Hello World!</body></note>
4	NULL	'char2'	?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">to>Us</to><from>Them</from><heading>note3</heading> <body>Hello World!</body></note>

Example 1

The following command exports the contents of USER.T1 in Delimited ASCII (DEL) format to the file "/mypath/t1export.del". Because the XML TO and XMLFILE options are not specified, the XML documents contained in columns C2 and C4 are written to the same path as the main exported file "/mypath". The base name for these files is "t1export.del.xml". The XMLSAVESCHEMA option indicates that XML schema information is saved during the export procedure.

```
EXPORT TO /mypath/t1export.del OF DEL XMLSAVESCHEMA SELECT * FROM USER.T1
```

The exported file "/mypath/t1export.del" contains:

```
2,"<XDS FIL='t1export.del.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='t1export.del.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='t1export.del.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

The exported XML file "/mypath/t1export.del.001.xml" contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him
   </to><from>Her</from><heading>note2</heading><body>Hello World!
   </body></note><?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">
   <to>Us</to><from>Them</from>heading>note3</heading><body>
   Hello World!</body></note>
```

Example 2

The following command exports the contents of USER.T1 in DEL format to the file "t1export.del". XML documents contained in columns C2 and C4 are written to the path "/home/user/xmlpath". The XML files are named with the base name

″xmldocs″, with multiple exported XML documents written to the same XML file. The XMLSAVESCHEMA option indicates that XML schema information is saved during the export procedure.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs XMLSAVESCHEMA SELECT * FROM USER.T1
```

The exported DEL file ″/home/user/t1export.del″ contains:

```
2,"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='xmldocs.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='xmldocs.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

The exported XML file ″/home/user/xmlpath/xmldocs.001.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00">
   <to>Him</to><from>Her</from><heading>note2</heading><body>
   Hello World!</body></note><?xml version="1.0" encoding="UTF-8" ?>
   <note time="14:00:00"><to>Us</to><from>Them</from><heading>
   note3</heading><body>Hello World!</body></note>
```

Example 3

The following command is similar to Example 2, except that each exported XML document is written to a separate XML file.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLINSEPFILES XMLSAVESCHEMA
SELECT * FROM USER.T1
```

The exported file ″/mypath/t1export.del″ contains:

```
2,"<XDS FIL='xmldocs.001.xml' />","char1","XDS FIL='xmldocs.002.xml' />"
4,,"char2","<XDS FIL='xmldocs.004.xml' SCH='S1.SCHEMA_A' />"
```

The exported XML file ″/home/user/xmlpath/xmldocs.001.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note>
```

The exported XML file ″/home/user/xmlpath/xmldocs.002.xml″ contains:

```
?xml version="1.0" encoding="UTF-8" ?>note time="13:00:00">to>Him/to>
   from>Her/from>heading>note2/heading>body>Hello World!/body>
   /note>
```

The exported XML file ″/home/user/xmlpath/xmldocs.004.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to>
   <from>Them</from><heading>note3</heading><body>Hello World!</body>
   </note>
```

Example 4

The following command writes the result of an XQuery to an XML file.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLNODECLARATION select
xmlquery( '$m/note/from/text()' passing by ref c4 as "m" returning sequence)
   from USER.T1
```

The exported DEL file ″/mypath/t1export.del″ contains:

```
"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='3' />"
"<XDS FIL='xmldocs.001.xml' OFF='3' LEN='4' />"
```

The exported XML file "/home/user/xmlpath/xmldocs.001.xml" contains:

```
HerThem
```

Note: The result of this particular XQuery does not produce well-formed XML documents. Therefore, the file exported above could not be directly imported into an XML column.

LBAC-protected data export considerations

When you export data that is protected by label-based access control (LBAC), the data that is exported is limited to the data that your LBAC credentials allow you to read.

If your LBAC credentials do not allow you to read a row, that row is not exported, but no error is returned. If your LBAC credentials do not allow you to read a column, the export utility fails, and an error (SQLSTATE 42512) is returned.

A value from a column with a data type of DB2SECURITYLABEL is exported as raw data enclosed in character delimiters. If a character delimiter is included in the original data, it is doubled. No other changes are made to the bytes that make up the exported value. This means that a data file that contains DB2SECURITYLABEL data can contain newlines, formfeeds, or other non-printable ASCII characters.

If you want the values of columns with a data type of DB2SECURITYLABEL to be exported in a human-readable form, you can use the SECLABEL_TO_CHAR scalar function in the SELECT statement to convert the values to the security label string format.

Examples

In the following examples, output is in DEL format and is written to the file myfile.del. The data is exported from a table named REPS, which was created with the following statement:

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

This example exports the values of the row_label column in the default format:

```
db2 export to myfile.del of del select * from reps
```

The data file is not very readable in most text editors because the values for the row_label column are likely to contain several ASCII control characters.

The following example exports the values of the row_label column in the security label string format:

```
db2 export to myfile.del of del select SECLABEL_TO_CHAR
(row_label,'DATA_ACCESS_POLICY'), id, name from reps
```

Here is an excerpt of the data file created by the previous example. Notice that the format of the security label is readable:

```
...
"Secret:():Epsilon 37", 2005, "Susan Liu"
"Secret:():(Epsilon 37,Megaphone,Cloverleaf)", 2006, "Johnny Cogent"
"Secret:():(Megaphone,Cloverleaf)", 2007, "Ron Imron"
...
```

Table export considerations

A typical export operation involves the outputting of selected data that is inserted or loaded into existing tables. However, it is also possible to export an entire table for subsequent re-creation using the import utility.

To export a table, you must specify the PC/IXF file format. You can then re-create your saved table (including its indexes) using the import utility in CREATE mode. However, some information is not saved to the exported IXF file if any of the following conditions exist:

- The index column names contain hexadecimal values of 0x2B or 0x2D.
- The table contains XML columns.
- The table is multidimensional clustered (MDC).
- The table contains a table partitioning key.
- The index name is longer than 128 bytes due to code page conversion.
- The table is protected.
- The EXPORT command contains action strings other than SELECT * FROM *tablename*
- You specify the **METHOD N** parameter for the export utility.

For a list of table attributes that are lost, see ″Table import considerations.″ If any information is not saved, warning SQL27984W is returned when the table is re-created.

Note: Import's CREATE mode is being deprecated. Use the db2look utility to capture and re-create your tables.

Index information
If the column names specified in the index contain either - or + characters, the index information is not collected, and warning SQL27984W is returned. The export utility completes its processing, and the data exported is unaffected. However, the index information is not saved in the IXF file. As a result, you must create the indexes separately using the db2look utility.

Space limitations
The export operation fails if the data that you are exporting exceeds the space available on the file system on which the exported file is created. In this case, you should limit the amount of data selected by specifying conditions on the WHERE clause so that the exported file fits on the target file system. You can run the export utility multiple times to export all of the data.

Tables with other file formats
If you do not export using the IXF file format, the output files do not contain descriptions of the target table, but they contain the record data. To re-create a table and its data, create the target table, then use the load or import utility to populate the table. You can use the db2look utility to capture the original table definitions and to generate the corresponding data definition language (DDL).

Typed table export considerations

You can use the DB2 export utility can be used to move data out of typed tables for a later import. Export moves data from one hierarchical structure of typed tables to another by following a specific order and creating an intermediate flat file.

When working with typed tables, the export utility controls what is placed in the output file; specify only the target table name and, optionally, the WHERE clause. You can express subselect statements only by specifying the target table name and the WHERE clause. You cannot specify a fullselect or select-statement when exporting a hierarchy.

Preservation of hierarchies using traverse order
Typed tables can be in a hierarchy. There are several ways you can move data across hierarchies:

- Movement from one hierarchy to an identical hierarchy
- Movement from one hierarchy to a subsection of a larger hierarchy
- Movement from a subsection of a large hierarchy to a separate hierarchy

Identification of types in a hierarchy is database dependent, meaning that in different databases, the same type has a different identifier. Therefore, when moving data between these databases, a mapping of the same types must be done to ensure that the data is moved correctly.

The mapping used for typed tables is known as the *traverse order*, the order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy. Before each typed row is written out during an export operation, an identifier is translated into an index value. This index value can be any number from one to the number of relevant types in the hierarchy. Index values are generated by numbering each type when moving through the hierarchy in a specific order—the traverse order. Figure 1 shows a hierarchy with four valid traverse orders:

- Person, Employee, Manager, Architect, Student
- Person, Student, Employee, Manager, Architect
- Person, Employee, Architect, Manager, Student
- Person, Student, Employee, Architect, Manager

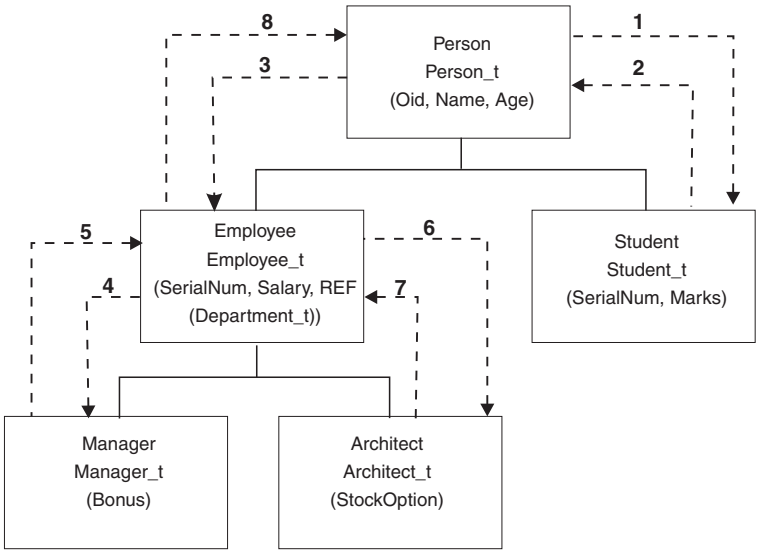


Figure 1. An example of a hierarchy

The traverse order is important when moving data between table hierarchies because it determines where the data is moved in relation to other data. There are two types of traverse order: *default* and *user specified*.

Default traverse order
With the default traverse order, all relevant types refer to all reachable types in the hierarchy from a given starting point in the hierarchy. The default order includes all tables in the hierarchy, and each table is ordered by the scheme used in the OUTER order predicate. For instance, the default traverse order of Figure 1, indicated by the dotted line, would be Person, Student, Employee, Manager, Architect.

The default traverse order behaves differently when used with different file formats. Exporting data to the PC/IXF file format creates a record of all relevant types, their definitions, and relevant tables. The export utility also completes the mapping of an index value to each table. When working with the PC/IXF file format, you should use the default traverse order.

With the ASC, DEL, or WSF file format, the order in which the typed rows and the typed tables are created could be different, even though the source and target hierarchies might be structurally identical. This results in time differences that the default traverse order identifies when proceeding through the hierarchies. The creation time of each type determines the order used to move through the hierarchy at both the source and the target when using the default traverse order. Ensure that the creation order of each type in both the source and the target hierarchies is identical and that there is structural identity between the source and the target. If these conditions cannot be met, select a user-specified traverse order.

User-specified traverse order
With the user-specified traverse order, you define (in a traverse order list) the relevant types to be used. This order outlines how to traverse the hierarchy and what sub-tables to export, whereas with the default traverse order, all tables in the hierarchy are exported.

Although you determine the starting point and the path down the hierarchy when defining the traverse order, remember that the subtables must be traversed in

pre-order fashion. Each branch in the hierarchy must be traversed to the bottom before a new branch can be started. The export utility looks for violations of this condition within the specified traverse order. One method of ensuring that the condition is met is to proceed from the top of the hierarchy (or the root table), down the hierarchy (subtables) to the bottom subtable, then back up to its supertable, down to the next ″right-most″ subtable, then back up to next higher supertable, down to its subtables, and so on.

If you want to control the traverse order through the hierarchies, ensure that the same traverse order is used for both the export and the import utilities.

Example 1

The following examples are based on the hierarchical structure in Figure 1. To export the entire hierarchy, enter the following commands:

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.ixf OF IXF HIERARCHY STARTING Person
```

Note that setting the parameter **HIERARCHY STARTING** to Person indicates that the default traverse order starting from the table PERSON.

Example 2

To export the entire hierarchy, but only the data for those people over the age of 20, you would enter the following commands:

```
DB2 CONNECT TO Source_db
   DB2 EXPORT TO entire_hierarchy.del OF DEL HIERARCHY (Person,
      Employee, Manager, Architect, Student) WHERE Age>=20
```

Note that setting the parameter **HIERARCHY** to Person, Employee, Manager, Architect, Student indicates a user-specified traverse order.

Identity column export considerations

You can use the export utility to export data from a table containing an identity column. However, the identity column limits your choice of output file format.

If the SELECT statement that you specify for the export operation is of the form SELECT * FROM *tablename* and you do not use the METHOD option is not used, exporting identity column properties to IXF files is supported. You can then use the REPLACE_CREATE and the CREATE options of the IMPORT command to re-create the table, including its identity column properties. If you create the exported IXF file from a table containing an identity column of type GENERATED ALWAYS, the only way that you can successfully import the data file is to specify the `identityignore` file type modifier during the import operation. Otherwise, all rows are rejected (SQL3550W is issued).

Note: The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.

LOB export considerations

When exporting tables with large object (LOB) columns, the default action is to export a maximum of 32 KB per LOB value and to place it in the same file as the rest of the column data. If you are exporting LOB values that exceed 32 KB, you should have the LOB data written to a separate file to avoid truncation.

To specify that LOB should be written to its own file, use the lobsinfile file type modifier. This modifier instructs the export utility to place the LOB data in the directories specified by the LOBS TO clause. Using LOBS TO or LOBFILE implicitly activates the lobsinfile file type modifier. By default, LOB values are written to the same path to which the exported relational data is written. If one or more paths are specified with the LOBS TO option, the export utility cycles between the paths to write each successful LOB value to the appropriate LOB file. You can also specify names for the output LOB files using the LOBFILE option. If the LOBFILE option is specified, the format of lobfilename is lobfilespec.xxx.lob, where lobfilespec is the value specified for the LOBFILE option, and xxx is a sequence number for lob files produced by the export utility. Otherwise, lobfilename is of the format: exportfilename.xxx.lob, where exportfilename is the name of the exported output file specified for the EXPORT command, and xxx is a sequence number for lob files produced by the export utility.

By default, LOBs are written to a single file, but you can also specify that the individual LOBs are to be stored in separate files. The export utility generates a LOB Location Specifier (LLS) to enable the storage of multiple LOBs in one file. The LLS, which is written to the export output file, is a string that indicates where the LOB data is stored within the file. The format of the LLS is lobfilename.nnn.mmm/, where lobfilename.ext is the name of the file that contains the LOB, nnn is the offset of the LOB within the file (measured in bytes), and mmm is the length of the LOB (measured in bytes). For example, an LLS of db2exp.001.123.456/ indicates that the LOB is located in the file db2exp.001, begins at an offset of 123 bytes into the file, and is 456 bytes long. If the indicated size in the LLS is 0, the LOB is considered to have a length of 0. If the length is -1, the LOB is considered to be NULL and the offset and file name are ignored.

If you don't want individual LOB data concatenated to the same file, use the lobsinsepfiles file type modifier to write each LOB to a separate file.

Note: The IXF file format does not store the LOB options of the column, such as whether or not the LOB column is logged. This means that the import utility cannot re-create a table containing a LOB column that is defined to be 1 GB or larger.

Example 1 The following example shows how to export LOBs (where the exported lob files have the specified base name lobs1) to a DEL file:

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1 modified by lobsinfile
   select * from emp_photo
```

Example 2 The following example shows how to export LOBs to a DEL file, where each LOB value is written to a separate file and lobfiles are written to two directories:

```
db2 export to myfile.del of del
lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
select * from emp_photo
```

Reference - Export

EXPORT

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Quick link to "File type modifiers for the export utility" on page 22.

Authorization

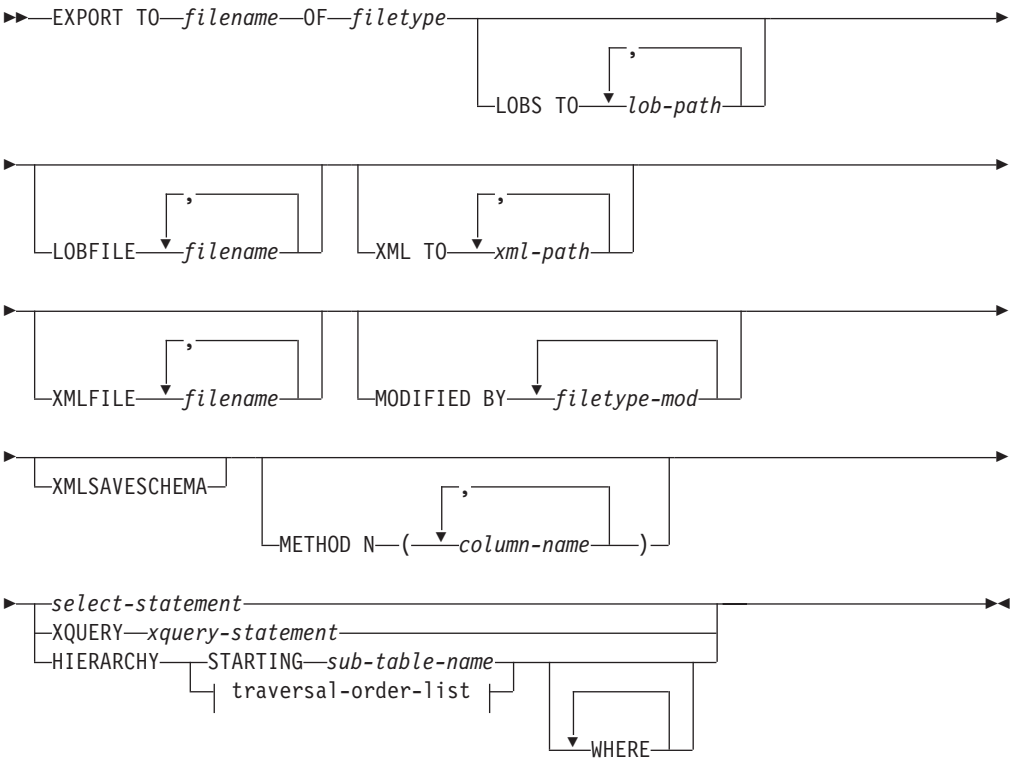
One of the following:

- *sysadm*
- *dbadm*

or CONTROL or SELECT privilege on each participating table or view.

Required connection

Command syntax



traversal-order-list:

Command parameters

HIERARCHY *traversal-order-list*

Export a sub-hierarchy using the specified traverse order. All sub-tables must be listed in PRE-ORDER fashion. The first sub-table name is used as the target table name for the SELECT statement.

HIERARCHY STARTING *sub-table-name*

Using the default traverse order (OUTER order for ASC, DEL, or WSF files, or the order stored in PC/IXF data files), export a sub-hierarchy starting from *sub-table-name*.

LOBFILE *filename*

Specifies one or more base file names for the LOB files. When name space is exhausted for the first name, the second name is used, and so on. The maximum number of file names that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *lob-path*), and then appending a 3-digit sequence number and the three character identifier `lob`. For example, if the current LOB path is the directory `/u/foo/lob/path/`, and the current LOB file name is `bar`, the LOB files created will be `/u/foo/lob/path/bar.001.lob`, `/u/foo/lob/path/bar.002.lob`, and so on.

LOBS TO *lob-path*

Specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

METHOD N *column-name*

Specifies one or more column names to be used in the output file. If this parameter is not specified, the column names in the table are used. This parameter is valid only for WSF and IXF files, but is not valid when exporting hierarchical data.

MODIFIED BY *filetype-mod*

Specifies file type modifier options. See "File type modifiers for the export utility" on page 22.

OF *filetype*

Specifies the format of the data in the output file:

- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs.
- WSF (work sheet format), which is used by programs such as:
  - Lotus® 1-2-3®
  - Lotus Symphony

  When exporting BIGINT or DECIMAL data, only values that fall within the range of type DOUBLE can be exported accurately. Although values that do not fall within this range are also exported, importing or loading these values back might result in incorrect data, depending on the operating system.
- IXF (Integration Exchange Format, PC version) is a proprietary binary format.

select-statement

Specifies the SELECT or XQUERY statement that will return the data to be

exported. If the statement causes an error, a message is written to the message file (or to standard output). If the error code is one of SQL0012W, SQL0347W, SQL0360W, SQL0437W, or SQL1824W, the export operation continues; otherwise, it stops.

TO *filename*

If the name of a file that already exists is specified, the export utility overwrites the contents of the file; it does not append the information.

XMLFILE *filename*

Specifies one or more base file names for the XML files. When name space is exhausted for the first name, the second name is used, and so on.

When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *xml-path*), appending a 3-digit sequence number, and appending the three character identifier `xml`. For example, if the current XML path is the directory `/u/foo/xml/path/`, and the current XML file name is `bar`, the XML files created will be `/u/foo/xml/path/bar.001.xml`, `/u/foo/xml/path/bar.002.xml`, and so on.

XML TO *xml-path*

Specifies one or more paths to directories in which the XML files are to be stored. There will be at least one file per XML path, and each file will contain at least one XQuery Data Model (XDM) instance. If more than one path is specified, then XDM instances are distributed evenly among the paths.

XMLSAVESCHEMA

Specifies that XML schema information should be saved for all XML columns. For each exported XML document that was validated against an XML schema when it was inserted, the fully qualified SQL identifier of that schema will be stored as an (SCH) attribute inside the corresponding XML Data Specifier (XDS). If the exported document was not validated against an XML schema or the schema object no longer exists in the database, an SCH attribute will not be included in the corresponding XDS.

The schema and name portions of the SQL identifier are stored as the ″OBJECTSCHEMA″ and ″OBJECTNAME″ values in the row of the SYSCAT.XSROBJECTS catalog table corresponding to the XML schema.

The XMLSAVESCHEMA option is not compatible with XQuery sequences that do not produce well-formed XML documents.

Usage notes

- Be sure to complete all table operations and release all locks before starting an export operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.
- Table aliases can be used in the SELECT statement.
- The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.
- The export utility produces a warning message whenever a character column with a length greater than 254 is selected for export to DEL format files.
- PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

- The file copying step is not necessary if the source and the target databases are both accessible from the same client.
- DB2 Connect can be used to export tables from DRDA® servers such as DB2 for OS/390®, DB2 for VM and VSE, and DB2 for OS/400®. Only PC/IXF export is supported.
- When exporting to the IXF format, if identifiers exceed the maximum size supported by the IXF format, the export will succeed but the resulting datafile cannot be used by a subsequent import operation using the CREATE mode. SQL27984W will be returned.
- When exporting to a diskette on Windows, and the table that has more data than the capacity of a single diskette, the system will prompt for another diskette, and multiple-part PC/IXF files (also known as multi-volume PC/IXF files, or logically split PC/IXF files), are generated and stored in separate diskettes. In each file, with the exception of the last, there is a DB2 CONTINUATION RECORD (or "AC" Record in short) written to indicate the files are logically split and where to look for the next file. The files can then be transferred to an AIX® system, to be read by the import and load utilities. The export utility will not create multiple-part PC/IXF files when invoked from an AIX system. For detailed usage, see the IMPORT command or LOAD command.
- The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form `SELECT * FROM tablename`.
- When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the WHERE clause. Fullselect and *select-statement* cannot be specified when exporting a hierarchy.
- For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.
- Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.
- When exporting data from a table that has protected rows, the LBAC credentials held by the session authorization id might limit the rows that are exported. Rows that the session authorization ID does not have read access to will not be exported. No error or warning is given.
- If the LBAC credentials held by the session authorization id do not allow reading from one or more protected columns included in the export then the export fails and an error (SQLSTATE 42512) is returned.
- Export packages are bound using `DATETIME ISO` format, thus, all date/time/timestamp values are converted into ISO format when cast to a string representation. Since the CLP packages are bound using `DATETIME LOC` format (locale specific format), you may see inconsistent behavior between CLP and export if the CLP DATETIME format is different from ISO. For instance, the following SELECT statement may return expected results:

```
db2 select col2 from tab1 where char(col2)='05/10/2005';
   COL2
   ----------
   05/10/2005
   05/10/2005
   05/10/2005
   3 record(s) selected.
```

  But an export command using the same select clause will not:

```
db2 export to test.del of del select col2 from test
where char(col2)='05/10/2005';
   Number of rows exported: 0
```

Now, replacing the LOCALE date format with ISO format gives the expected results:

```
db2 export to test.del of del select col2 from test
where char(col2)='2005-05-10';
   Number of rows exported: 3
```

File type modifiers for the export utility

Table 3. Valid file type modifiers for the export utility: All file formats

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file `db2exp.001`, and is 456 bytes long. If you specify the "lobsinfile" modifier when using EXPORT, the LOB data is placed in the locations specified by the LOBS TO clause. Otherwise the LOB data is sent to the data file directory. The LOBS TO clause specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The LOBS TO or LOBFILE options will implicitly activate the LOBSINFILE behavior. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
xmlinsepfiles	Each XQuery Data Model (XDM) instance is written to a separate file. By default, multiple values are concatenated together in the same file.
lobsinsepfiles	Each LOB value is written to a separate file. By default, multiple values are concatenated together in the same file.
xmlnodeclaration	XDM instances are written without an XML declaration tag. By default, XDM instances are exported with an XML declaration tag at the beginning that includes an encoding attribute.
xmlchar	XDM instances are written in the character codepage. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, XDM instances are written out in Unicode.
xmlgraphic	If the `xmlgraphic` modifier is specified with the EXPORT command, the exported XML document will be encoded in the UTF-16 code page regardless of the application code page or the `codepage` file type modifier.

Table 4. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[2] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	The single quotation mark (′) can also be specified as a character string delimiter as follows:
	`modified by chardel''`

codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data to this code page from the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[2]
	In the following example, `coldel;` causes the export utility to use the semicolon character (;) as a column delimiter for the exported data:
	`db2 "export to temp of del modified by coldel;`
	`select * from staff where dept = 20"`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[2]
nochardel	Column data will not be surrounded by character delimiters. This option should not be specified if the data is intended to be imported or loaded using DB2. It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.[2]

striplzeros	Removes the leading zeros from all exported decimal columns.
	Consider the following example:
	`db2 create table decimalTable ( c1 decimal( 31, 2 ) )`
	`db2 insert into decimalTable values ( 1.1 )`
	`db2 export to data of del select * from decimalTable`
	`db2 export to data of del modified by STRIPLZEROS`
	`select * from decimalTable`
	In the first export operation, the content of the exported file data will be +0000000000000000000000000001.10. In the second operation, which is identical to the first except for the `striplzeros` modifier, the content of the exported file data will be +1.10.

Table 4. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[4] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements)`
	`UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microseond elements)`
	`UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microseond elements)`
	`UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microseond elements)`
	`UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microseond elements)`
	`U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microseond elements)`
	`TT - Meridian indicator (AM or PM)`
	Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The MMM element will produce the following values: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', and 'Dec'. 'Jan' is equal to month 1, and 'Dec' is equal to month 12.
	The following example illustrates how to export data containing user-defined time stamp formats from a table called 'schedule':
	`db2 export to delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`select * from schedule`

Table 5. Valid file type modifiers for the export utility: IXF file format

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data from this code page to the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

Table 6. Valid file type modifiers for the export utility: WSF file format

Modifier	Description
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[5] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[5]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[5]
4	Creates a WSF file containing DBCS characters.

Note:

1. The export utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the export operation fails, and an error code is returned.
2. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
3. The export utility normally writes
   - date data in *YYYYMMDD* format
   - char(date) data in *"YYYY-MM-DD"* format
   - time data in *"HH.MM.SS"* format
   - time stamp data in *"YYYY-MM-DD-HH. MM.SS.uuuuuu"* format

   Data contained in any datetime columns specified in the SELECT statement for the export operation will also be in these formats.
4. For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```
5. These files can also be directed to a specific product by specifying an `L` for Lotus 1-2-3, or an `S` for Symphony in the *filetype-mod* parameter string. Only one value or product designator can be specified.

6. The WSF file format is not supported for XML columns.
7. All XDM instances are written to XML files that are separate from the main data file, even if neither the XMLFILE nor the XML TO clause is specified. By default, XML files are written to the path of the exported data file. The default base name for XML files is the name of the exported data file with the extension ″.xml″ appended to it.
8. All XDM instances are written with an XML declaration at the beginning that includes an encoding attribute, unless the XMLNODECLARATION file type modifier is specified.
9. By default, all XDM instances are written in Unicode unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. The default path for XML data and LOB data is the path of the main data file. The default XML file base name is the main data file. The default LOB file base name is the main data file. For example, if the main data file is
    `/mypath/myfile.del`

    , the default path for XML data and LOB data is
    `/mypath"`

    , the default XML file base name is
    `myfile.del`

    , and the default LOB file base name is
    `myfile.del`

    .
    The LOBSINFILE file type modifier must be specified in order to have LOB files generated.
11. The export utility appends a numeric identifier to each LOB file or XML file. The identifier starts as a 3 digit, 0 padded sequence value, starting at
    `.001`

    . After the 999th LOB file or XML file, the identifier will no longer be padded with zeroes (for example, the 1000th LOG file or XML file will have an extension of
    `.1000`

    . Following the numeric identifier is a three character type identifier representing the data type, either
    `.lob`

    or
    `.xml`

    . For example, a generated LOB file would have a name in the format
    `myfile.del.001.lob`

    , and a generated XML file would be have a name in the format
    `myfile.del.001.xml`

    .
12. It is possible to have the export utility export XDM instances that are not well-formed documents by specifying an XQuery. However, you will not be

able to import or load these exported documents directly into an XML column, since XML columns can only contain complete documents.

EXPORT command using the ADMIN_CMD procedure

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Quick link to "File type modifiers for the export utility" on page 31.

Authorization

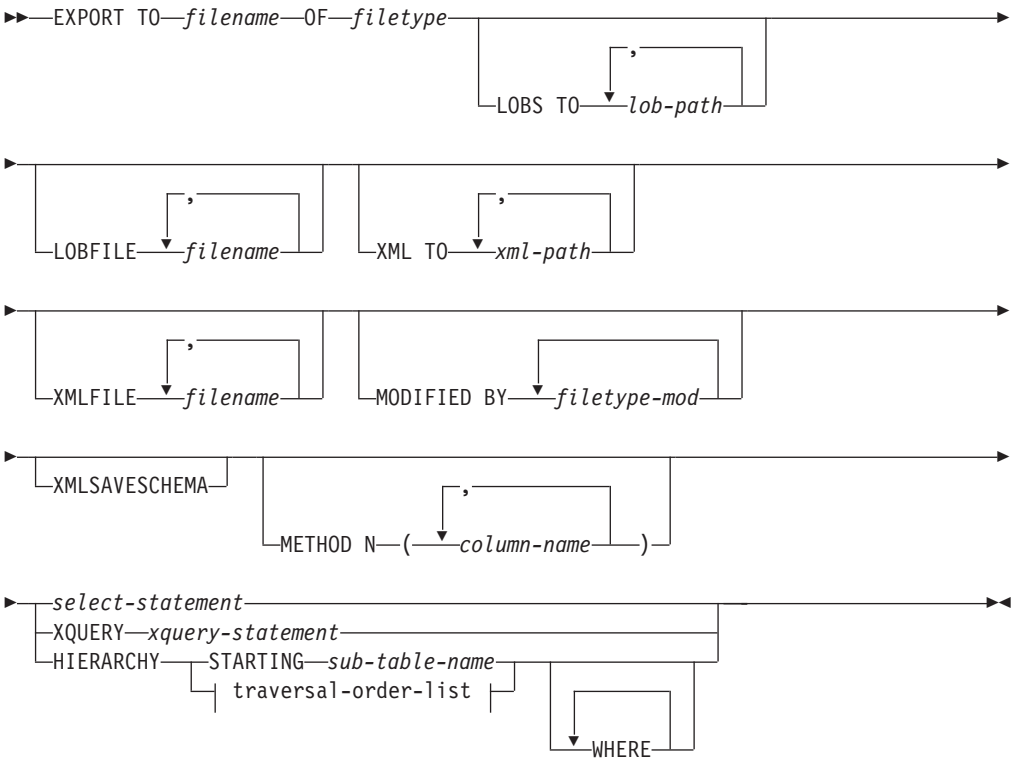
One of the following:

- *sysadm*
- *dbadm*

or CONTROL or SELECT privilege on each participating table or view.

Required connection

Command syntax



traversal-order-list:

Command parameters

HIERARCHY *traversal-order-list*
Export a sub-hierarchy using the specified traverse order. All sub-tables must be listed in PRE-ORDER fashion. The first sub-table name is used as the target table name for the SELECT statement.

HIERARCHY STARTING *sub-table-name*
Using the default traverse order (OUTER order for ASC, DEL, or WSF files, or the order stored in PC/IXF data files), export a sub-hierarchy starting from *sub-table-name*.

LOBFILE *filename*
Specifies one or more base file names for the LOB files. When name space is exhausted for the first name, the second name is used, and so on. The maximum number of file names that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *lob-path*), and then appending a 3-digit sequence number and the three character identifier `lob`. For example, if the current LOB path is the directory `/u/foo/lob/path/`, and the current LOB file name is `bar`, the LOB files created will be `/u/foo/lob/path/bar.001.lob`, `/u/foo/lob/path/bar.002.lob`, and so on.

LOBS TO *lob-path*
Specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

METHOD N *column-name*
Specifies one or more column names to be used in the output file. If this parameter is not specified, the column names in the table are used. This parameter is valid only for WSF and IXF files, but is not valid when exporting hierarchical data.

MODIFIED BY *filetype-mod*
Specifies file type modifier options. See "File type modifiers for the export utility" on page 31.

OF *filetype*
Specifies the format of the data in the output file:

- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs.
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony

  When exporting BIGINT or DECIMAL data, only values that fall within the range of type DOUBLE can be exported accurately. Although values that do not fall within this range are also exported, importing or loading these values back might result in incorrect data, depending on the operating system.
- IXF (Integration Exchange Format, PC version) is a proprietary binary format.

select-statement
Specifies the SELECT or XQUERY statement that will return the data to be

exported. If the statement causes an error, a message is written to the message file (or to standard output). If the error code is one of SQL0012W, SQL0347W, SQL0360W, SQL0437W, or SQL1824W, the export operation continues; otherwise, it stops.

TO *filename*

If the name of a file that already exists is specified, the export utility overwrites the contents of the file; it does not append the information.

XMLFILE *filename*

Specifies one or more base file names for the XML files. When name space is exhausted for the first name, the second name is used, and so on.

When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *xml-path*), appending a 3-digit sequence number, and appending the three character identifier `xml`. For example, if the current XML path is the directory `/u/foo/xml/path/`, and the current XML file name is `bar`, the XML files created will be `/u/foo/xml/path/bar.001.xml`, `/u/foo/xml/path/bar.002.xml`, and so on.

XML TO *xml-path*

Specifies one or more paths to directories in which the XML files are to be stored. There will be at least one file per XML path, and each file will contain at least one XQuery Data Model (XDM) instance. If more than one path is specified, then XDM instances are distributed evenly among the paths.

XMLSAVESCHEMA

Specifies that XML schema information should be saved for all XML columns. For each exported XML document that was validated against an XML schema when it was inserted, the fully qualified SQL identifier of that schema will be stored as an (SCH) attribute inside the corresponding XML Data Specifier (XDS). If the exported document was not validated against an XML schema or the schema object no longer exists in the database, an SCH attribute will not be included in the corresponding XDS.

The schema and name portions of the SQL identifier are stored as the ″OBJECTSCHEMA″ and ″OBJECTNAME″ values in the row of the SYSCAT.XSROBJECTS catalog table corresponding to the XML schema.

The XMLSAVESCHEMA option is not compatible with XQuery sequences that do not produce well-formed XML documents.

Usage notes

- Be sure to complete all table operations and release all locks before starting an export operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.
- Table aliases can be used in the SELECT statement.
- The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.
- The export utility produces a warning message whenever a character column with a length greater than 254 is selected for export to DEL format files.
- PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

- The file copying step is not necessary if the source and the target databases are both accessible from the same client.
- DB2 Connect can be used to export tables from DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF export is supported.
- When exporting to the IXF format, if identifiers exceed the maximum size supported by the IXF format, the export will succeed but the resulting datafile cannot be used by a subsequent import operation using the CREATE mode. SQL27984W will be returned.
- When exporting to a diskette on Windows, and the table that has more data than the capacity of a single diskette, the system will prompt for another diskette, and multiple-part PC/IXF files (also known as multi-volume PC/IXF files, or logically split PC/IXF files), are generated and stored in separate diskettes. In each file, with the exception of the last, there is a DB2 CONTINUATION RECORD (or "AC" Record in short) written to indicate the files are logically split and where to look for the next file. The files can then be transferred to an AIX system, to be read by the import and load utilities. The export utility will not create multiple-part PC/IXF files when invoked from an AIX system. For detailed usage, see the IMPORT command or LOAD command.
- The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form `SELECT * FROM tablename`.
- When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the WHERE clause. Fullselect and *select-statement* cannot be specified when exporting a hierarchy.
- For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.
- Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.
- When exporting data from a table that has protected rows, the LBAC credentials held by the session authorization id might limit the rows that are exported. Rows that the session authorization ID does not have read access to will not be exported. No error or warning is given.
- If the LBAC credentials held by the session authorization id do not allow reading from one or more protected columns included in the export then the export fails and an error (SQLSTATE 42512) is returned.
- Export packages are bound using `DATETIME ISO` format, thus, all date/time/timestamp values are converted into ISO format when cast to a string representation. Since the CLP packages are bound using `DATETIME LOC` format (locale specific format), you may see inconsistent behavior between CLP and export if the CLP DATETIME format is different from ISO. For instance, the following SELECT statement may return expected results:

```
db2 select col2 from tab1 where char(col2)='05/10/2005';
   COL2
   ----------
   05/10/2005
   05/10/2005
   05/10/2005
   3 record(s) selected.
```

But an export command using the same select clause will not:

```
db2 export to test.del of del select col2 from test
where char(col2)='05/10/2005';
   Number of rows exported: 0
```

Now, replacing the LOCALE date format with ISO format gives the expected results:

```
db2 export to test.del of del select col2 from test
where char(col2)='2005-05-10';
   Number of rows exported: 3
```

File type modifiers for the export utility

Table 7. Valid file type modifiers for the export utility: All file formats

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file `db2exp.001`, and is 456 bytes long. If you specify the "lobsinfile" modifier when using EXPORT, the LOB data is placed in the locations specified by the LOBS TO clause. Otherwise the LOB data is sent to the data file directory. The LOBS TO clause specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The LOBS TO or LOBFILE options will implicitly activate the LOBSINFILE behavior. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
xmlinsepfiles	Each XQuery Data Model (XDM) instance is written to a separate file. By default, multiple values are concatenated together in the same file.
lobsinsepfiles	Each LOB value is written to a separate file. By default, multiple values are concatenated together in the same file.
xmlnodeclaration	XDM instances are written without an XML declaration tag. By default, XDM instances are exported with an XML declaration tag at the beginning that includes an encoding attribute.
xmlchar	XDM instances are written in the character codepage. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, XDM instances are written out in Unicode.
xmlgraphic	If the `xmlgraphic` modifier is specified with the EXPORT command, the exported XML document will be encoded in the UTF-16 code page regardless of the application code page or the `codepage` file type modifier.

Table 8. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[2] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	The single quotation mark (') can also be specified as a character string delimiter as follows:
	`modified by chardel''`

codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data to this code page from the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[2]
	In the following example, `coldel;` causes the export utility to use the semicolon character (;) as a column delimiter for the exported data:
	`db2 "export to temp of del modified by coldel;`
	`select * from staff where dept = 20"`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[2]
nochardel	Column data will not be surrounded by character delimiters. This option should not be specified if the data is intended to be imported or loaded using DB2. It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.[2]

striplzeros	Removes the leading zeros from all exported decimal columns.
	Consider the following example:
	`db2 create table decimalTable ( c1 decimal( 31, 2 ) )`
	`db2 insert into decimalTable values ( 1.1 )`
	`db2 export to data of del select * from decimalTable`
	`db2 export to data of del modified by STRIPLZEROS`
	`select * from decimalTable`
	In the first export operation, the content of the exported file data will be +0000000000000000000000000000001.10. In the second operation, which is identical to the first except for the `striplzeros` modifier, the content of the exported file data will be +1.10.

Table 8. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[4] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements)`
	`UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microseond elements)`
	`UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microseond elements)`
	`UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microseond elements)`
	`UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microseond elements)`
	`U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microseond elements)`
	`TT - Meridian indicator (AM or PM)`
	Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The MMM element will produce the following values: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', and 'Dec'. 'Jan' is equal to month 1, and 'Dec' is equal to month 12.
	The following example illustrates how to export data containing user-defined time stamp formats from a table called 'schedule':
	`db2 export to delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`select * from schedule`

Table 9. Valid file type modifiers for the export utility: IXF file format

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data from this code page to the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

Table 10. Valid file type modifiers for the export utility: WSF file format

Modifier	Description
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[5] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[5]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[5]
4	Creates a WSF file containing DBCS characters.

Note:

1. The export utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the export operation fails, and an error code is returned.
2. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
3. The export utility normally writes
   - date data in *YYYYMMDD* format
   - char(date) data in *"YYYY-MM-DD"* format
   - time data in *"HH.MM.SS"* format
   - time stamp data in *"YYYY-MM-DD-HH. MM.SS.uuuuuu"* format

   Data contained in any datetime columns specified in the SELECT statement for the export operation will also be in these formats.
4. For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```
5. These files can also be directed to a specific product by specifying an `L` for Lotus 1-2-3, or an `S` for Symphony in the *filetype-mod* parameter string. Only one value or product designator can be specified.

6. The WSF file format is not supported for XML columns.
7. All XDM instances are written to XML files that are separate from the main data file, even if neither the XMLFILE nor the XML TO clause is specified. By default, XML files are written to the path of the exported data file. The default base name for XML files is the name of the exported data file with the extension ".xml" appended to it.
8. All XDM instances are written with an XML declaration at the beginning that includes an encoding attribute, unless the XMLNODECLARATION file type modifier is specified.
9. By default, all XDM instances are written in Unicode unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. The default path for XML data and LOB data is the path of the main data file. The default XML file base name is the main data file. The default LOB file base name is the main data file. For example, if the main data file is
    `/mypath/myfile.del`

    , the default path for XML data and LOB data is
    `/mypath"`

    , the default XML file base name is
    `myfile.del`

    , and the default LOB file base name is
    `myfile.del`

    .
    The LOBSINFILE file type modifier must be specified in order to have LOB files generated.
11. The export utility appends a numeric identifier to each LOB file or XML file. The identifier starts as a 3 digit, 0 padded sequence value, starting at
    `.001`

    . After the 999th LOB file or XML file, the identifier will no longer be padded with zeroes (for example, the 1000th LOG file or XML file will have an extension of
    `.1000`

    . Following the numeric identifier is a three character type identifier representing the data type, either
    `.lob`

    or
    `.xml`

    . For example, a generated LOB file would have a name in the format
    `myfile.del.001.lob`

    , and a generated XML file would be have a name in the format
    `myfile.del.001.xml`

    .
12. It is possible to have the export utility export XDM instances that are not well-formed documents by specifying an XQuery. However, you will not be

able to import or load these exported documents directly into an XML column, since XML columns can only contain complete documents.

db2Export - Export data from a database

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Authorization

One of the following:

- sysadm
- dbadm

or CONTROL or SELECT privilege on each participating table or view. Label-based access control (LBAC) is enforced for this function. The data that is exported may be limited by the LBAC credentials of the caller if the data is protected by LBAC.

Required connection

Database. If implicit connect is enabled, a connection to the default database is established.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Export (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ExportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqlu_media_list *piLobFileList;
   struct sqldcol *piDataDescriptor;
   struct sqllob *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ExportOut *poExportInfoOut;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2ExportStruct;

typedef SQL_STRUCTURE db2ExportIn
{
   db2Uint16 *piXmlSaveSchema;
} db2ExportIn;

typedef SQL_STRUCTURE db2ExportOut
{
   db2Uint64 oRowsExported;
} db2ExportOut;
```

```
SQL_API_RC SQL_API_FN
  db2gExport (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gExportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqlu_media_list *piLobFileList;
   struct sqldcol *piDataDescriptor;
   struct sqllob *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ExportOut *poExportInfoOut;
   db2Uint16 iDataFileNameLen;
   db2Uint16 iFileTypeLen;
   db2Uint16 iMsgFileNameLen;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2gExportStruct;
```

db2Export API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2ExportStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ExportStruct data structure parameters

piDataFileName
Input. A string containing the path and the name of the external file into which the data is to be exported.

piLobPathList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files are to be stored. Exported LOB data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piLobFileList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its sqlu_location_entry structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on. When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from piLobPathList), and then appending a 3-digit sequence number and the .lob extension. For example, if the current LOB path is the directory /u/foo/lob/path, the current LOB file name is bar, and the LOBSINSEPFILES file type modifier is set, then

the created LOB files will be /u/foo/LOB/path/bar.001.lob, /u/foo/LOB/path/bar.002.lob, and so on. If the LOBSINSEPFILES file type modifier is not set, then all the LOB documents will be concatenated and put into one file /u/foo/lob/path/bar.001.lob

piDataDescriptor
Input. Pointer to an sqldcol structure specifying the column names for the output file. The value of the dcolmeth field determines how the remainder of the information provided in this parameter is interpreted by the export utility. Valid values for this parameter (defined in sqlutil header file, located in the include directory) are:

SQL_METH_N
Names. Specify column names to be used in the output file.

SQL_METH_D
Default. Existing column names from the table are to be used in the output file. In this case, the number of columns and the column specification array are both ignored. The column names are derived from the output of the SELECT statement specified in piActionString.

piActionString
Input. Pointer to an sqllob structure containing a valid dynamic SQL SELECT statement. The structure contains a 4-byte long field, followed by the characters that make up the SELECT statement. The SELECT statement specifies the data to be extracted from the database and written to the external file.

The columns for the external file (from piDataDescriptor), and the database columns from the SELECT statement, are matched according to their respective list/structure positions. The first column of data selected from the database is placed in the first column of the external file, and its column name is taken from the first element of the external column array.

piFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in sqlutil header file) are:

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM® Personal Decision Series programs, and many other database managers and file managers.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

SQL_IXF
PC version of the Integration Exchange Format, the preferred method for exporting data from a table. Data exported to this file format can later be imported or loaded into the same table or into another database manager table.

piFileTypeMod
Input. A pointer to an sqldcol structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link below: "File type modifiers for the export utility."

piMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, the information is appended . If it does not exist, a file is created.

iCallerAction
Input. An action requested by the caller. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API. If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested export operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

poExportInfoOut
A pointer to the db2ExportOut structure.

piExportInfoIn
Input. Pointer to the db2ExportIn structure.

piXmlPathList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files are to be stored. Exported XML data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piXmlFileList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its sqlu_location_entry structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on. When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from piXmlFileList), and then appending a 3-digit sequence number and the .xml extension. For example, if the current XML path is the directory /u/foo/xml/path, the current XML file name is bar, and the XMLINSEPFILES file type modifier is set,

then the created XML files will be /u/foo/xml/path/bar.001.xml, /u/foo/xml/path/bar.002.xml, and so on. If the XMLINSEPFILES file type modifier is not set, then all the XML documents will be concatenated and put into one file /u/foo/xml/path/bar.001.xml

db2ExportIn data structure parameters

piXmlSaveSchema
Input. Indicates that the SQL identifier of the XML schema used to validate each exported XML document should be saved in the exported data file. Possible values are TRUE and FALSE.

db2ExportOut data structure parameters

oRowsExported
Output. Returns the number of records exported to the target file.

db2gExportStruct data structure specific parameters

iDataFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the data file name.

iFileTypeLen
Input. A 2-byte unsigned integer representing the length in bytes of the file type.

iMsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

Usage notes

Before starting an export operation, you must complete all table operations and release all locks in one of two ways:

- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

Table aliases can be used in the SELECT statement.

The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.

If the export utility produces warnings, the message will be written out to a message file, or standard output if one is not specified.

A warning message is issued if the number of columns (dcolnum field of sqldcol structure) in the external column name array, piDataDescriptor, is not equal to the number of columns generated by the SELECT statement. In this case, the number of columns written to the external file is the lesser of the two numbers. Excess database columns or external column names are not used to generate the output file.

If the db2uexpm.bnd module or any other shipped .bnd files are bound manually, the format option on the binder must not be used.

DB2 Connect can be used to export tables from DRDA servers such as DB2 for z/OS® and OS/390, DB2 for VM and VSE, and DB2 for System i™. Only PC/IXF export is supported.

PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

The export utility will not create multiple-part PC/IXF files when invoked from an AIX system.

Index definitions for a table are included in the PC/IXF file when the contents of a single database table are exported to a PC/IXF file with a piActionString parameter beginning with SELECT * FROM tablename, and the piDataDescriptor parameter specifying default names. Indexes are not saved for views, or if the SELECT clause of the piActionString includes a join. A WHERE clause, a GROUP BY clause, or a HAVING clause in the piActionString parameter will not prevent the saving of indexes. In all of these cases, when exporting from typed tables, the entire hierarchy must be exported.

The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form: SELECT * FROM tablename.

When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the WHERE clause. Fullselect and select-statement cannot be specified when exporting a hierarchy.

For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.

Note: Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.

REXX™ API syntax

```
EXPORT :stmt TO datafile OF filetype
[MODIFIED BY :filetmod] [USING :dcoldata]
MESSAGES msgfile [ROWS EXPORTED :number]


CONTINUE EXPORT


STOP EXPORT
```

REXX API parameters

stmt A REXX host variable containing a valid dynamic SQL SELECT statement. The statement specifies the data to be extracted from the database.

datafile
Name of the file into which the data is to be exported.

filetype
The format of the data in the export file. The supported file formats are:

DEL Delimited ASCII

WSF Worksheet format

IXF PC version of Integration Exchange Format.

filetmod
A host variable containing additional processing options.

dcoldata
A compound REXX host variable containing the column names to be used in the export file. In the following, XXX represents the name of the host variable:

XXX.0 Number of columns (number of elements in the remainder of the variable).

XXX.1 First column name.

XXX.2 Second column name.

XXX.3 and so on.

If this parameter is NULL, or a value for dcoldata has not been specified, the utility uses the column names from the database table.

msgfile
File, path, or device name where error and warning messages are to be sent.

number
A host variable that will contain the number of exported rows.

Export sessions - CLP examples

Example 1
The following example shows how to export information from the STAFF table in the SAMPLE database (to which the user must be connected) to myfile.ixf, with the output in IXF format. If the database connection is not through DB2 Connect, the index definitions (if any) will be stored in the output file; otherwise, only the data will be stored:

```
db2 export to myfile.ixf of ixf messages msgs.txt select * from staff
```

Example 2
The following example shows how to export the information about employees in Department 20 from the STAFF table in the SAMPLE database (to which the user must be connected) to awards.ixf, with the output in IXF format:

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff
   where dept = 20
```

Example 3 The following example shows how to export LOBs to a DEL file:

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1, lobs2 modified by lobsinfile
   select * from emp_photo
```

Example 4
The following example shows how to export LOBs to a DEL file, specifying a second directory for files that might not fit into the first directory:

```
db2 export to myfile.del of del
   lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
   select * from emp_photo
```

Example 5 The following example shows how to export data to a DEL file, using a single quotation mark as the string delimiter, a semicolon as the column delimiter, and a comma as the decimal point. The same convention should be used when importing data back into the database:

```
db2 export to myfile.del of del
   modified by chardel'' coldel; decpt,
   select * from staff
```

Chapter 3. Import utility

Import overview

The import utility populates a table, typed table, or view with data using an SQL INSERT statement. If the table or view receiving the imported data already contains data, the input data can either replace or be appended to the existing data.

Like export, import is a relatively simple data movement utility. It can be activated through the Control Center, by issuing CLP commands, by calling the ADMIN_CMD stored procedure, or by calling its API, db2Import, through a user application.

There are a number of data formats that import supports, as well as features that can be used with import:

- Import supports IXF, WSF, ASC, and DEL data formats.
- Import can be used with file type modifiers to customize the import operation.
- Import can be used to move hierarchical data and typed tables.
- Import logs all activity, updates indexes, verifies constraints, and fires triggers.
- Import allows you to specify the names of the columns within the table or view into which the data is to be inserted.
- Import can be used with DB2 Connect.

Import modes

Import has five modes which determine the method in which the data is imported. The first three, INSERT, INSERT_UPDATE, and REPLACE are used when the target tables already exist. All three support IXF, WSF, ASC, and DEL data formats. However, only INSERT and INSERT_UPDATE can be used with nicknames.

Table 11.

Mode	Best practice usage
INSERT	Inserts input data into target table without changing existing data
INSERT_UPDATE	Updates rows with matching primary key values with values of input rows Where there's no matching row, inserts imported row into the table
REPLACE	Deletes all existing data and inserts imported data, while keeping table and index definitions

The other two modes, REPLACE_CREATE and CREATE, are used when the target tables do not exist. They can only be used with input files in the PC/IXF format, which contains a structured description of the table that is to be created. Imports cannot be performed in these modes if the object table has any dependents other than itself.

© Copyright IBM Corp. 1993, 2008

Note: Import's CREATE and REPLACE_CREATE modes are being deprecated. Use the db2look utility instead.

Table 12.

Mode	Best practice usage
REPLACE_CREATE	Deletes all existing data and inserts imported data, while keeping table and index definitions Creates target table and index if they don't exist
CREATE	Creates target table and index Can specify the name of the table space where the new table is created

How import works

The number of steps and the amount of time required for an import depend on the amount of data being moved and the options that you specify. An import operation follows these steps:

1. Locking tables
   Import acquires either an exclusive (X) lock or a nonexclusive (IX) lock on existing target tables, depending on whether you allow concurrent access to the table.
2. Locating and retrieving data
   Import uses the FROM clause to locate the input data. If your command indicates that XML or LOB data is present, import will locate this data.
3. Inserting data
   Import either replaces existing data or adds new rows of data to the table.
4. Checking constraints and firing triggers
   As the data is written, import ensures that each inserted row complies with the constraints defined on the target table. Information about rejected rows is written to the messages file. Import also fires existing triggers.
5. Committing the operation
   Import saves the changes made and releases the locks on the target table. You can also specify that periodic take place during the import.

The following items are mandatory for a basic import operation:

- The path and the name of the input file
- The name or alias of the target table or view
- The format of the data in the input file
- The method by which the data is to be imported
- The traverse order, when importing hierarchical data
- The subtable list, when importing typed tables

Additional options
There are a number of options that allow you to customize an import operation. You can specify file type modifiers in the MODIFIED BY clause to change the format of the data, tell the import utility what to do with the data, and to improve performance.

The import utility, by default, does not perform commits until the end of a successful import, except in the case of some ALLOW WRITE ACCESS imports.

This improves the speed of an import, but for the sake of concurrency, restartability, and active log space considerations, it might be preferable to specify that commits take place during the import. One way of doing so is to set the **COMMITCOUNT** parameter to "automatic," which instructs import to internally determine when it should perform a commit. Alternatively, you can set **COMMITCOUNT** to a specific number, which instructs import to perform a commit once that specified number of records has been imported.

There are a few ways to improve import's performance. As the import utility is an embedded SQL application and does SQL fetches internally, optimizations that apply to SQL operations apply to import as well. You can use the `compound` file type modifier to perform a specified number of rows to insert at a time, rather than the default row-by-row insertion. If you anticipate that a large number of warnings will be generated (and, therefore, slow down the operation) during the import, you can also specify the `norowwarnings` file type modifier to suppress warnings about rejected rows.

Messages file
During an import, standard ASCII text message files are written to contain the error, warning, and informational messages associated with that operation. If the utility is invoked through the application programming interface (API) db2Import, you must specify the name of these files in advance with the **MESSAGES** parameter, otherwise it is optional. The messages file is a convenient way of monitoring the progress of an import, as you can access is while the import is in progress. In the event of a failed import operation, message files can be used to determine a restarting point by indicating the last row that was successfully imported.

Note: If the volume of output messages generated by an import operation against a remote database exceeds 60 KB, the utility will keep the first 30 KB and the last 30 KB.

Privileges and authorities required to use import

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

With SYSADM or DBADM authority, you can perform any type of import operation. The table below lists the other authorities on each participating table, view or nickname that enable you to perform the corresponding type of import.

Table 13.

Mode	Required authority
INSERT	CONTROL or INSERT and SELECT
INSERT_UPDATE	CONTROL or INSERT, SELECT, UPDATE, and DELETE
REPLACE	CONTROL or INSERT, SELECT, and DELETE

Table 13. (continued)

Mode	Required authority
REPLACE_CREATE	When the target table exists: CONTROL or INSERT, SELECT, and DELETE When the target table doesn't exist: CREATETAB (on the database), USE (on the table space), and when the schema does not exist: IMPLICIT_SCHEMA (on the database), or when the schema exists: CREATEIN (on the schema)
CREATE	CREATETAB (on the database), USE (on the table space), and when the schema does not exist: IMPLICIT_SCHEMA (on the database), or when the schema exists: CREATEIN (on the schema)

Note: The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.
As well, to use the REPLACE or REPLACE_CREATE option on a table, the session authorization ID must have the authority to drop the table.

If you want to import to a hierarchy, the required authority also depends on the mode. For existing hierarchies, CONTROL privilege on every subtable in the hierarchy is sufficient for a REPLACE operation. For hierarchies that don't exist, CONTROL privilege on every subtable in the hierarchy, along with CREATETAB and USE, is sufficient for a REPLACE_CREATE operation.

In addition, there a few considerations for importing into tables with label-based access control (LBAC) security labels defined on them. To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. To import data into a table that has protected rows, the session authorization ID must have been granted a security label for write access that is part of the security policy protecting the table.

Importing data

The import utility inserts data from an external file with a supported file format into a table, hierarchy, view or nickname. The load utility is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Before invoking the import utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be imported. If implicit connect is enabled, a connection to the default database is established. Utility access to DB2 for Linux, UNIX, or Windows database servers from DB2 for Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect gateway or loop back environment. Since the utility will issue a COMMIT or a ROLLBACK statement, you should complete all transactions and release all locks by issuing a COMMIT statement or a ROLLBACK operation before invoking import.

Note: The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.

The following restrictions apply to the import utility:

- If the existing table is a parent table containing a primary key that is referenced by a foreign key in a dependent table, its data cannot be replaced, only appended to.
- You cannot perform an import replace operation into an underlying table of a materialized query table defined in refresh immediate mode.
- You cannot import data into a system table, a summary table, or a table with a structured type column.
- You cannot import data into declared temporary tables.
- Views cannot be created through the import utility.
- Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using SELECT *.)
- Because the import utility generates its own SQL statements, the maximum statement size of 2 MB might, in some cases, be exceeded.
- You cannot re-create a partitioned table or an multidimensional clustered table (MDC) using the CREATE or REPLACE_CREATE import options.
- You cannot re-create tables containing XML columns.
- You cannot import encrypted data.
- The import replace operation does not honor the Not Logged Initially clause. The IMPORT command's REPLACE option does not honor the CREATE TABLE statement's NOT LOGGED INITIALLY (NLI) clause or the ALTER TABLE statement's ACTIVATE NOT LOGGED INITIALLY clause. If an import with the REPLACE action is performed within the same transaction as a CREATE TABLE or ALTER TABLE statement where the NLI clause is invoked, the import will not honor the NLI clause. All inserts will be logged.

  Workaround 1: Delete the contents of the table using the DELETE statement, then invoke the import with INSERT statement.

  Workaround 2: Drop the table and re-create it, then invoke the import with INSERT statement.

The following limitation applies to the import utility: If the volume of output messages generated by an import operation against a remote database exceeds 60 KB, the utility will keep the first 30 KB and the last 30 KB.

The import utility can be invoked through the command line processor (CLP), the Import Table notebook in the Control Center, or by calling the application programming interface (API) db2Import from a client application.

Using the Import Table notebook

1. From the Control Center, expand the object tree until you find the Tables folder.
2. Click on the Tables folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane).
3. Right-click on the table you want in the contents pane, and select Import from the pop-up menu. The Import Table notebook opens.

Detailed information about the Import Table notebook is provided through the Control Center online help facility.

Issuing IMPORT commands through the CLP

A very simple import operation requires you to specify only an input file, a file format, an import mode, and a target table (or the name of the table that is to be created).

For example, to import data from the CLP, enter the IMPORT command:

```
db2 import from filename of fileformat import_mode into table
```

where `filename` is the name of the input file that contains the data you want to import, `ixf` is the file format, `insert` is the mode, and `table` is the name of the table that you want to insert the data into.

However, you might also want to specify a messages file to which warning and error messages will be written. To do that, add the **MESSAGES** parameter and a message file name so the command is:

```
db2 import from filename of fileformat messages messagefile import_mode into table
```

For complete syntax and usage information, see "IMPORT command."

Importing XML data

The import utility can be used to import XML data into an XML table column using either the table name or a nickname for a DB2 Database for Linux, UNIX, and Windows source data object.

When importing data into an XML table column, you can use the XML FROM option to specify the paths of the input XML data file or files. For example, for an XML file "/home/user/xmlpath/xmldocs.001.xml" that had previously been exported, the following command could be used to import the data back into the table.

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

Validating inserted documents against schemas

The XMLVALIDATE option allows XML documents to be validated against XML schemas as they are imported. In the following example, incoming XML documents are validated against schema information that was saved when the XML documents were exported:

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T1
```

Specifying parse options

You can use the XMLPARSE option to specify whether whitespace in the imported XML documents is preserved or stripped. In the following example, all imported XML documents are validated against XML schema information that was saved when the XML documents were exported, and these documents are parsed with whitespace preserved.

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING XDS INSERT INTO USER.T1
```

Imported table re-creation

You can use the import utility's CREATE mode to re-create a table that was saved through the export utility. However, there are a number of limitations on the process, as many of the input table's attributes are not retained.

For import to be able to re-create the table, the export operation must meet some requirements. The original table must have been exported to an IXF file. If you export files with DEL or ASC file formats, the output files do not contain descriptions of the target table, but they contain the record data. To re-create a table with data stored in these file formats, create the target table, then use the load or import utility to populate the table from these files. You can use the db2look utility to capture the original table definitions and to generate the corresponding data definition language (DDL). As well, the SELECT statement used during the export can only contain certain action strings. For example, no column names can be used in the SELECT clause and only SELECT * is permitted.

Note: Import's CREATE mode is being deprecated. Use the db2look utility to capture and re-create your tables.

Retained attributes
The re-created table will retain the following attributes of the original table:

- The primary key name, and definition
- Column information, including:
  - Column name
  - Column data type, including user-defined distinct types, which are preserved as their base type
  - Identity properties
  - Lengths (except for lob_file types)
  - Code page (if applicable)
  - Identity options
  - Whether the column is defined as nullable or not nullable
  - Default values for constants, if any, but not other types of default values
- Index information, including:
  - Index name
  - Index creator name
  - Column names, and whether each column is sorted in ascending or descending order
  - Whether the index is defined as unique
  - Whether the index is clustered
  - Whether the index allows reverse scans
  - PCTFREE values
  - MINPCTUSED values

Note: No index information is retained if the column names in the index contain the characters - or +, in which case SQL27984W is returned.

Lost attributes
The re-created table does not retain several attributes of the original table, including:

- Whether the source was a normal table, a materialized query table (MQT), a view, or a set of columns from any or all of these sources
- Unique constraints and other types of constraints or triggers (not including primary key constraints)
- Table information, including:
  - MQT definition (if applicable)

  - MQT options (if applicable)
  - Table space options; however, this information can be specified through the IMPORT command
  - Multidimensional clustering (MDC) dimensions
  - Partitioned table dimensions
  - Table partitioning key
  - NOT LOGGED INITIALLY property
  - Check constraints
  - Table code page
  - Protected table properties
  - Table or value compression options
- Column information, including:
  - Any default value except constant values
  - LOB options (if any)
  - XML properties
  - References clause of the CREATE TABLE statement (if any)
  - Referential constraints (if any)
  - Check constraints (if any)
  - Generated column options (if any)
  - Columns dependent on database scope sequences
- Index information, including:
  - INCLUDE columns (if any)
  - Index name, if the index is a primary key index
  - Descending order of keys, if the index is a primary key index (ascending is the default)
  - Index column names that contain hexadecimal values of 0x2B or 0x2D
  - Index names that contain more than 128 bytes after code page conversion
  - PCTFREE2 value
  - Unique constraints

Note: This list is not exhaustive, use with care.

If the import fails and SQL3311N is returned, you can still re-create the table using the file type modifier `forcecreate`. This modifier allows you to create the table with missing or limited information.

Typed table import considerations

The import utility can be used to move data both from and into typed tables while preserving the data's preexisting hierarchy. If desired, import can also be used to create the table hierarchy and the type hierarchy.

The movement of data from one hierarchical structure of typed tables to another is done through a specific traverse order and the creation of an intermediate flat file during an export operation. In turn, the import utility controls the size and the placement of the hierarchy being moved, using the CREATE, INTO *table-name*, UNDER, and AS ROOT TABLE parameters. As well, import determines what is placed in the target database. For example, it can specify an attributes list at the

end of each subtable name to restrict the attributes that are moved to the target database. If no attributes list is used, all of the columns in each subtable are moved.

Table re-creation
The type of import you are able to perform depends on the file format of the input file. When working with ASC, DEL, or WSF data, the target table or hierarchy must exist before the data can be imported. However, data from a PC/IXF file can be imported even if the table or hierarchy does not already exist if you specify an import CREATE operation. It must be noted that if the CREATE option is specified, import cannot alter subtable definitions.

Traverse order
The traverse order contained in the input file enables the hierarchies in the data to be maintained. Therefore, the same traverse order must be used when invoking the export utility and the import utility.

For the PC/IXF file format, one need only specify the target subtable name, and use the default traverse order stored in the file.

When using options other than CREATE with typed tables, the traverse order list enables one to specify the traverse order. This user-specified traverse order must match the one used during the export operation. The import utility guarantees the accurate movement of data to the target database given the following:

- An identical definition of subtables in both the source and the target databases
- An identical hierarchical relationship among the subtables in both the source and target databases
- An identical traverse order

Although you determine the starting point and the path down the hierarchy when defining the traverse order, each branch must be traversed to the end before the next branch in the hierarchy can be started. The import utility looks for violations of this condition within the specified traverse order.

Examples

Examples in this section are based on the following hierarchical structure with four valid traverse orders:

- Person, Employee, Manager, Architect, Student
- Person, Student, Employee, Manager, Architect
- Person, Employee, Architect, Manager, Student
- Person, Student, Employee, Architect, Manager

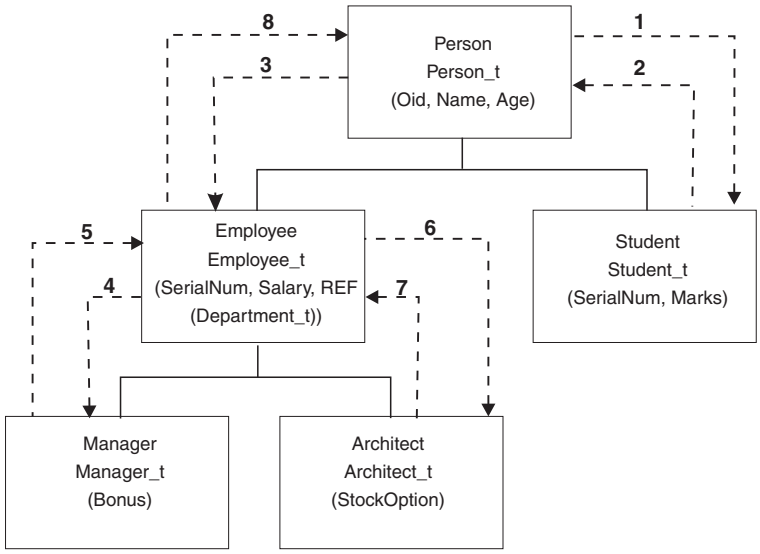


Figure 2. An example of a hierarchy

Example 1
To re-create an entire hierarchy (contained in the data file entire_hierarchy.ixf created by a prior export operation) using import, you would enter the following commands:

```
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.ixf OF IXF CREATE INTO
   HIERARCHY STARTING Person AS ROOT TABLE
```

Each type in the hierarchy is created if it does not exist. If these types already exist, they must have the same definition in the target database as in the source database. An SQL error (SQL20013N) is returned if they are not the same. Since a new hierarchy is being created, none of the subtables defined in the data file being moved to the target database (Target_db) can exist. Each of the tables in the source database hierarchy is created. Data from the source database is imported into the correct subtables of the target database.

Example 2
To re-create the entire hierarchy of the source database and import it to the target database, while only keeping selected data, you would enter the following commands:

```
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.del OF DEL INSERT INTO (Person,
   Employee(Salary), Architect) IN HIERARCHY (Person, Employee,
   Manager, Architect, Student)
```

The target tables PERSON, EMPLOYEE, and ARCHITECT must all exist. Data is imported into the PERSON, EMPLOYEE, and ARCHITECT subtables. That is, the following will be imported:

- All columns in PERSON into PERSON
- All columns in PERSON plus SALARY in EMPLOYEE into EMPLOYEE
- All columns in PERSON plus SALARY in EMPLOYEE, plus all columns in ARCHITECT into ARCHITECT

Columns SerialNum and REF(Employee_t) are not imported into EMPLOYEE or its subtables (that is, ARCHITECT, which is the only subtable having data imported into it).

Note: Because ARCHITECT is a subtable of EMPLOYEE, and the only import column specified for EMPLOYEE is SALARY, SALARY is also the only Employee-specific column imported into ARCHITECT. That is, neither SerialNum nor REF(Employee_t) columns are imported into either EMPLOYEE or ARCHITECT rows.
Data for the MANAGER and the STUDENT tables is not imported.

Example 3
This example shows how to export from a regular table, and import as a single subtable in a hierarchy. The EXPORT command operates on regular (non-typed) tables, so there is no Type_id column in the data file. The file type modifier `no_type_id` is used to indicate this, so that the import utility does not expect the first column to be the Type_id column.

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO Student_sub_table.del OF DEL SELECT * FROM
   Regular_Student
DB2 CONNECT TO Target_db
DB2 IMPORT FROM Student_sub_table.del OF DEL METHOD P(1,2,3,5,4)
   MODIFIED BY NO_TYPE_ID INSERT INTO HIERARCHY (Student)
```

In this example, the target table STUDENT must exist. Since STUDENT is a subtable, the modifier `no_type_id` is used to indicate that there is no Type_id in the first column. However, you must ensure that there is an existing Object_id column, in addition to all of the other attributes that exist in the STUDENT table. Object-id is expected to be the first column in each row imported into the STUDENT table. The METHOD clause reverses the order of the last two attributes.

LBAC-protected data import considerations

For a successful import operation into a table with protected rows, you must have LBAC (label-based access control) credentials. You must also provide a valid security label, or a security label that can be converted to a valid label, for the security policy currently associated with the target table.

If you do not have valid LBAC credentials, the import fails and an error (SQLSTATE 42512) is returned. In cases where the input data does not contain a security label or that security label is not in its internal binary format, you can use several file type modifiers to allow your import to proceed.

When you import data into a table with protected rows, the target table has one column with a data type of DB2SECURITYLABEL. If the input row of data does not contain a value for that column, that row is rejected unless the `usedefaults` file type modifier is specified in the import command, in which case the security label you hold for write access from the security policy protecting the table is used. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

When you import data into a table that has protected rows and the input data does include a value for the column with a data type of DB2SECURITYLABEL, the same rules are followed as when you insert data into that table. If the security label protecting the row being imported (the one in that row of the data file) is one that you are able to write to, then that security label is used to protect the row. (In other words, it is written to the column that has a data type of

DB2SECURITYLABEL.) If you are not able to write to a row protected by that security label, what happens depends on how the security policy protecting the source table was created:

- If the CREATE SECURITY POLICY statement that created the policy included the option RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL, the insert fails and an error is returned.
- If the CREATE SECURITY POLICY statement did not include the option or if it instead included the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, the security label in the data file for that row is ignored and the security label you hold for write access is used to protect that row. No error or warning is issued in this case. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

Delimiter considerations

When importing data into a column with a data type of DB2SECURITYLABEL, the value in the data file is assumed by default to be the actual bytes that make up the internal representation of that security label. However, some raw data might contain newline characters which could be misinterpreted by the IMPORT command as delimiting the row. If you have this problem, use the `delprioritychar` file type modifier to ensure that the character delimiter takes precedence over the row delimiter. When you use `delprioritychar`, any record or column delimiters that are contained within character delimiters are not recognized as being delimiters. Using the `delprioritychar` file type modifier is safe to do even if none of the values contain a newline character, but it does slow the import down slightly.

If the data being imported is in ASC format, you might want to take an extra step in order to prevent any trailing white space from being included in the imported security labels and security label names. ASCII format uses column positions as delimiters, so this might occur when importing into variable-length fields. Use the `striptblanks` file type modifier to truncate any trailing blank spaces.

Nonstandard security label values

You can also import data files in which the values for the security labels are strings containing the values of the components in the security label, for example, *S:(ALPHA,BETA)*. To do so you must use the file type modifier `seclabelchar`. When you use `seclabelchar`, a value for a column with a data type of DB2SECURITYLABEL is assumed to be a string constant containing the security label in the string format for security labels. If a string is not in the proper format, the row is not inserted and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table, the row is not inserted and a warning (SQLSTATE 01H53) is returned.

You can also import a data file in which the values of the security label column are security label names. To import this sort of file you must use the file type modifier `seclabelname`. When you use `seclabelname`, all values for columns with a data type of DB2SECURITYLABEL are assumed to be string constants containing the names of existing security labels. If no security label exists with the indicated name for the security policy protecting the table, the row is not inserted and a warning (SQLSTATE 01H53) is returned.

Examples

For all examples, the input data file myfile.del is in DEL format. All are importing data into a table named REPS, which was created with this statement:

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

For this example, the input file is assumed to contain security labels in the default format:

```
db2 import from myfile.del of del modified by delprioritychar insert into reps
```

For this example, the input file is assumed to contain security labels in the security label string format:

```
db2 import from myfile.del of del modified by seclabelchar insert into reps
```

For this example, the input file is assumed to contain security labels names for the security label column:

```
db2 import from myfile.del of del modified by seclabelname insert into reps
```

Buffered-insert imports

In a partitioned database environment, the import utility can be enabled to use buffered inserts. This reduces the messaging that occurs when data is imported, resulting in better performance.

The buffered inserts option should only be enabled if you are not concerned about error reporting, since details about a failed buffered insert are not returned.

When buffered inserts are used, import sets a default **WARNINGCOUNT** value to 1. As a result, the operation will fail if any rows are rejected. If a record is rejected, the utility will roll back the current transaction. The number of committed records can be used to determine which records were successfully inserted into the database. The number of committed records can be non zero only if the COMMITCOUNT option was specified.

If a different **WARNINGCOUNT** value is explicitly specified on the import command, and some rows are rejected, the row summary output by the utility can be incorrect. This is due to a combination of the asynchronous error reporting used with buffered inserts and the fact that an error detected during the insertion of a group of rows causes all the rows of that group to be backed out. Since the utility would not reliably report which input records were rejected, it would be difficult to determine which records were committed and which records need to be re-inserted into the database.

Use the DB2 bind utility to request buffered inserts. The import package, db2uimpm.bnd, must be rebound against the database using the INSERT BUF option. For example:

```
db2 connect to your_database
db2 bind db2uimpm.bnd insert buf
```

Buffered inserts feature cannot be used in conjunction with import operations in the INSERT_UPDATE mode. The bind file db2uImpInsUpdate.bnd enforces this restriction. This file should never be bound with the INSERT BUF option. This

causes the import operations in the INSERT_UPDATE mode to fail. Import operations in the INSERT, REPLACE, or REPLACE_CREATE modes are not affected by the binding of the new file.

Identity column import considerations

The import utility can be used to import data into a table containing an identity column whether or not the input data has identity column values.

If no identity-related file type modifiers are used, the utility works according to the following rules:

- If the identity column is GENERATED ALWAYS, an identity value is generated for a table row whenever the corresponding row in the input file is missing a value for the identity column, or a NULL value is explicitly given. If a non-NULL value is specified for the identity column, the row is rejected (SQL3550W).
- If the identity column is GENERATED BY DEFAULT, the import utility makes use of user-supplied values, if they are provided; if the data is missing or explicitly NULL, a value is generated.

The import utility does not perform any extra validation of user-supplied identity values beyond what is normally done for values of the identity column's data type (that is, SMALLINT, INT, BIGINT, or DECIMAL). Duplicate values will not be reported. In addition, the `compound=x` modifier cannot be used when importing data into a table with an identity column.

There are two ways you can simplify the import of data into tables that contain an identity column: the `identitymissing` and the `identityignore` file type modifiers.

Importing data without an identity column

The `identitymissing` modifier makes importing a table with an identity column more convenient if the input data file does not contain any values (not even NULLS) for the identity column. For example, consider a table defined with the following SQL statement:

```
create table table1 (c1 char(30),
                     c2 int generated by default as identity,
                     c3 real,
                     c4 char(1))
```

A user might want to import data from a file (import.del) into TABLE1, and this data might have been exported from a table that does not have an identity column. The following is an example of such a file:

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

One way to import this file would be to explicitly list the columns to be imported through the IMPORT command as follows:

```
db2 import from import.del of del replace into table1 (c1, c3, c4)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of importing the file is to use the `identitymissing` file type modifier as follows:

```
db2 import from import.del of del modified by identitymissing
   replace into table1
```

Importing data with an identity column
The identityignore modifier is in some ways the opposite of the identitymissing modifier: it indicates to the import utility that even though the input data file contains data for the identity column, the data should be ignored, and an identity value should be generated for each row. For example, a user might want to import the following data from a file (import.del) into TABLE1, as defined above:

```
Robert, 1, 45.2, J
Mike, 2, 76.9, K
Leo, 3, 23.4, I
```

If the user-supplied values of 1, 2, and 3 are not to be used for the identity column, the user could issue the following IMPORT command:

```
db2 import from import.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The identityignore modifier simplifies the syntax as follows:

```
db2 import from import.del of del modified by identityignore
   replace into table1
```

When a table with an identity column is exported to an IXF file, the REPLACE_CREATE and the CREATE options of the IMPORT command can be used to re-create the table, including its identity column properties. If such an IXF file is created from a table containing an identity column of type GENERATED ALWAYS, the only way that the data file can be successfully imported is to specify the identityignore modifier. Otherwise, all rows will be rejected (SQL3550W).

Note: The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.

Generated column import considerations

The import utility can be used to import data into a table containing (nonidentity) generated columns whether or not the input data has generated column values.

If no generated column-related file type modifiers are used, the import utility works according to the following rules:

- A value is generated for a generated column whenever the corresponding row in the input file is missing a value for the column, or a NULL value is explicitly given. If a non-NULL value is supplied for a generated column, the row is rejected (SQL3550W).
- If the server generates a NULL value for a generated column that is not nullable, the row of data to which this field belongs is rejected (SQL0407N). This could happen, for example, if a non-nullable generated column were defined as the sum of two table columns that have NULL values supplied to them in the input file.

There are two ways you can simplify the import of data into tables that contain a generated column: the generatedmissing and the generatedignore file type modifiers.

Importing data without generated columns
The generatedmissing modifier makes importing data into a table with generated columns more convenient if the input data file does not contain any values (not even NULLS) for all generated columns present in the table. For example, consider a table defined with the following SQL statement:

```
create table table1 (c1 int,
                     c2 int,
                     g1 int generated always as (c1 + c2),
                     g2 int generated always as (2 * c1),
                     c3 char(1))
```

A user might want to import data from a file (load.del) into TABLE1, and this data might have been exported from a table that does not have any generated columns. The following is an example of such a file:

```
1, 5, J
2, 6, K
3, 7, I
```

One way to import this file would be to explicitly list the columns to be imported through the IMPORT command as follows:

```
db2 import from import.del of del replace into table1 (c1, c2, c3)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of importing the file is to use the `generatedmissing` file type modifier as follows:

```
db2 import from import.del of del modified by generatedmissing
   replace into table1
```

Importing data with generated columns

The `generatedignore` modifier is in some ways the opposite of the `generatedmissing` modifier: it indicates to the import utility that even though the input data file contains data for all generated columns, the data should be ignored, and values should be generated for each row. For example, a user might want to import the following data from a file (import.del) into TABLE1, as defined above:

```
1, 5, 10, 15, J
2, 6, 11, 16, K
3, 7, 12, 17, I
```

The user-supplied, non-NULL values of `10`, `11`, and `12` (for g1), and `15`, `16`, and `17` (for g2) result in the row being rejected (SQL3550W). To avoid this, the user could issue the following IMPORT command:

```
db2 import from import.del of del method P(1, 2, 5)
   replace into table1 (c1, c2, c3)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The `generatedignore` modifier simplifies the syntax as follows:

```
db2 import from import.del of del modified by generatedignore
   replace into table1
```

For an INSERT_UPDATE, if the generated column is also a primary key and the `generatedignore` modifier is specified, the IMPORT command honors the `generatedignore` modifier. The IMPORT command does not substitute the user-supplied value for this column in the WHERE clause of the UPDATE statement.

LOB import considerations

Since the import utility restricts the size of a single column value to 32 KB, extra considerations need to be taken when importing LOBs.

The import utility, by default, treats data in the input file as data to load into the column. However, when large object (LOB) data is stored in the main input data

file, the size of the data is limited to 32 KB. Therefore, to prevent loss of data, LOB data should be stored separate from the main datafile and the `lobsinfile` file type modifier should be specified when importing LOBs.

The LOBS FROM clause implicitly activates `lobsinfile`. The LOBS FROM clause conveys to the import utility the list of paths to search for the LOB files while importing the data. If LOBS FROM option is not specified, the LOB files to import are assumed to reside in the same path as the input relational data file.

Indicating where LOB data is stored
The LOB Location Specifier (LLS) can be used to store multiple LOBs in a single file when importing the LOB information. The export utility generates and stores it in the export output file when `lobsinfile` is specified, and it indicates where LOB data can be found. When data with the modified by `lobsinfile` option specified is being imported, the database will expect an LLS for each of the corresponding LOB columns. If something other than an LLS is encountered for a LOB column, the database will treat it as a LOB file and will load the entire file as the LOB.

For an import in CREATE mode, you can specify that the LOB data be created and stored in a separate table space by using the LONG IN clause.

The following example shows how you would import an DEL file which has its LOBs stored in separate files:

```
IMPORT FROM inputfile.del OF DEL
 LOBS FROM /tmp/data
 MODIFIED BY lobsinfile
 INSERT INTO newtable
```

User-defined distinct types import considerations

The import utility casts user-defined distinct types (UDTs) to similar base data types automatically. This saves you from having to explicitly cast UDTs to the base data types. Casting allows for comparisons between UDTs and the base data types in SQL.

Additional considerations for import

Client/server environments and import

When you import a file to a remote database, a stored procedure can be called to perform the import on the server.

A stored procedure cannot be called when:

- The application and database code pages are different.
- The file being imported is a multiple-part PC/IXF file.
- The method used for importing the data is either column name or relative column position.
- The target column list provided is longer than 4 KB.
- The LOBS FROM clause or the `lobsinfile` modifier is specified.
- The NULL INDICATORS clause is specified for ASC files.

When import uses a stored procedure, messages are created in the message file using the default language installed on the server. The messages are in the language of the application if the language at the client and the server are the same.

The import utility creates two temporary files in the tmp subdirectory of the sqllib directory (or the directory indicated by the **DB2INSTPROF** registry variable, if specified). One file is for data, and the other file is for messages generated by the import utility.

If you receive an error about writing or opening data on the server, ensure that:

- The directory exists.
- There is sufficient disk space for the files.
- The instance owner has write permission in the directory.

Table locking during import

The import utility supports two table locking modes: offline, or ALLOW NO ACCESS, mode; and online, or ALLOW WRITE ACCESS mode.

ALLOW NO ACCESS mode prevents concurrent applications from accessing table data. ALLOW WRITE ACCESS mode allows concurrent applications both read and write access to the import target table. If no mode is explicitly specified, import runs in the default mode, ALLOW NO ACCESS. As well, the import utility is, by default, bound to the database with isolation level RS (read stability).

Offline import (ALLOW NO ACCESS)

In ALLOW NO ACCESS mode, import acquires an exclusive (X) lock on the target table is before inserting any rows. Holding a lock on a table has two implications:

- First, if there are other applications holding a table lock or row locks on the import target table, the import utility waits for those applications to commit or roll back their changes.
- Second, while import is running, any other application requesting locks waits for the import operation to complete.

Note: You can specify a locktimeout value, which prevents applications (including the import utility) from waiting indefinitely for a lock.
By requesting an exclusive lock at the beginning of the operation, import prevents deadlocks from occurring as a result of other applications working and holding row locks on the same target table.

Online import (ALLOW WRITE ACCESS)

In ALLOW WRITE ACCESS mode, the import utility acquires a nonexclusive (IX) lock on the target table. Holding this lock on the table has the following implications:

- If there are other applications holding an incompatible table lock, the import utility does not start inserting data until all of these applications commit or roll back their changes.
- While import is running, any other application requesting an incompatible table lock waits until the import commits or rolls back the current transaction. Note that import's table lock does not persist across a transaction boundary. As a result, online import has to request and potentially wait for a table lock after every commit.
- If there are other applications holding an incompatible row lock, the import utility stops inserting data until all of these applications commit or roll back their changes.

- While import is running, any other application requesting an incompatible row lock waits until the import operation commits or rolls back the current transaction.

To preserve the online properties, and to reduce the chance of a deadlock, an ALLOW WRITE ACCESS import periodically commits the current transaction and releases all row locks before escalating to an exclusive table lock. If you have not explicitly set a commit frequency, import performs commits as if COMMITCOUNT AUTOMATIC has been specified. No commits are performed if COMMITCOUNT is set to 0.

ALLOW WRITE ACCESS mode is not compatible with the following:

- Imports in REPLACE, CREATE, or REPLACE_CREATE mode
- Imports with buffered inserts
- Imports into a target view
- Imports into a hierarchy table
- Imports into a table with its lock granularity is set at the table level (set by using the LOCKSIZE parameter of the ALTER TABLE statement)

Reference - Import

IMPORT

Inserts data from an external file with a supported file format into a table, hierarchy, view or nickname. LOAD is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Quick link to "File type modifiers for the import utility" on page 76.

Authorization

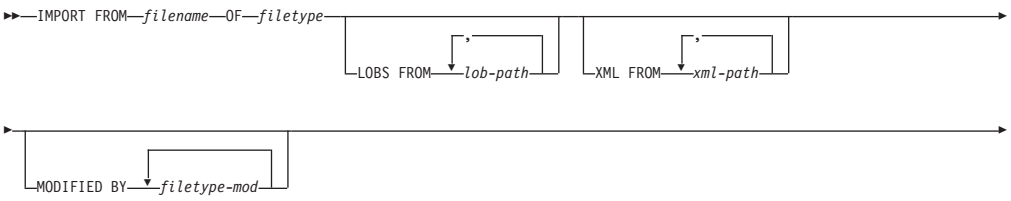
- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:

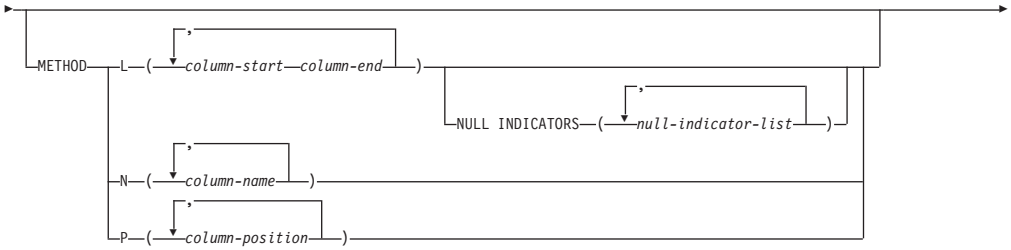
  - *sysadm*
  - *dbadm*
  - CREATETAB authority on the database and USE privilege on the table space, as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- IMPORT to a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CREATETAB authority on the database and USE privilege on the table space and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the REPLACE_CREATE option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the REPLACE option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on every sub-table in the hierarchy
- To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the import fails and an error (SQLSTATE 42512) is returned.
- To import data into a table that has protected rows, the session authorization ID must hold LBAC credentials that meet these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access

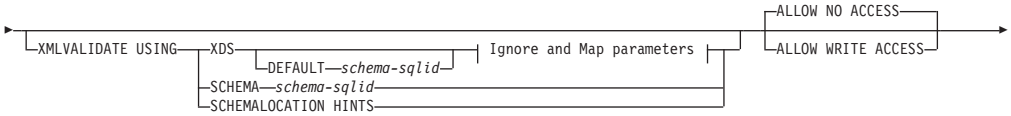
  The label on the row to insert, the user's LBAC credentials, the security policy definition, and the LBAC rules determine the label on the row.
- If the REPLACE or REPLACE_CREATE option is specified, the session authorization ID must have the authority to drop the table.

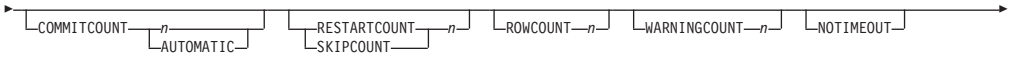
Required connection

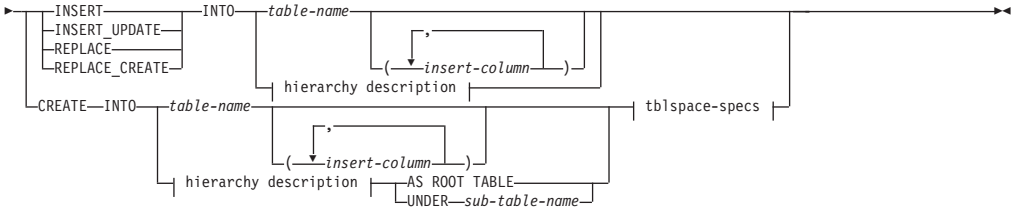
Command syntax



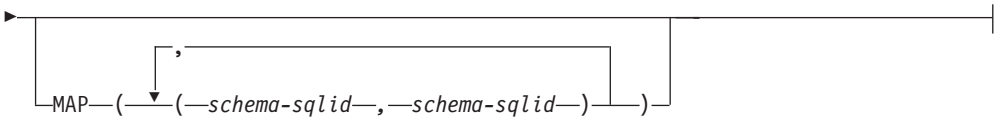






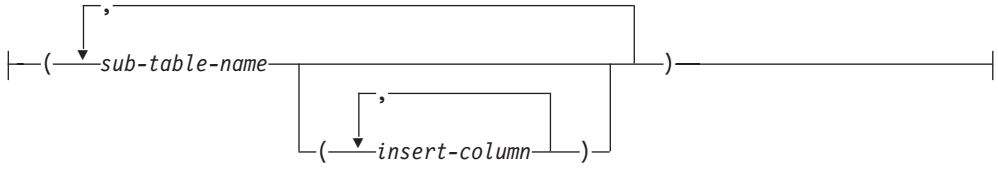


Ignore and Map parameters:



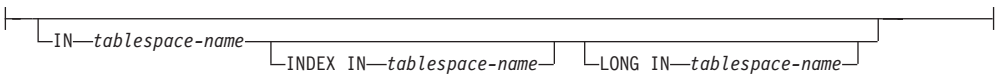
hierarchy description:

sub-table-list:



traversal-order-list:

tblspace-specs:



Command parameters

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal order.

ALLOW NO ACCESS
Runs import in the offline mode. An exclusive (X) lock on the target table is acquired before any rows are inserted. This prevents concurrent applications from accessing table data. This is the default import behavior.

ALLOW WRITE ACCESS
Runs import in the online mode. An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the REPLACE, CREATE, or REPLACE_CREATE import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the COMMITCOUNT option was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required when you import to a nickname and COMMITCOUNT must be specified with a valid number (AUTOMATIC is not considered a valid option).

AS ROOT TABLE
Creates one or more sub-tables as a stand-alone table hierarchy.

COMMITCOUNT *n* **| AUTOMATIC**
Performs a COMMIT after every *n* records are imported. When a number *n* is specified, import performs a COMMIT after every *n* records are imported. When compound inserts are used, a user-specified commit frequency of *n* is rounded up to the first integer multiple of the compound count value. When AUTOMATIC is specified, import internally determines when a commit needs to be performed. The utility will commit for either one of two reasons:

- to avoid running out of active log space
- to avoid lock escalation from row level to table level

If the ALLOW WRITE ACCESS option is specified, and the COMMITCOUNT option is not specified, the import utility will perform commits as if COMMITCOUNT AUTOMATIC had been specified.

If the IMPORT command encounters an SQL0964C (Transaction Log Full) while inserting or updating a record and the COMMITCOUNT *n* is specified, IMPORT will attempt to resolve the issue by performing an

unconditional commit and then reattempt to insert or update the record. If this does not help resolve the log full condition (which would be the case when the log full is attributed to other activity on the database), then the IMPORT command will fail as expected, however the number of rows committed may not be a multiple of the COMMITCOUNT *n* value. The RESTARTCOUNT or SKIPCOUNT option can be used to avoid processing those row already committed.

CREATE

Note: The CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

Creates the table definition and row contents in the code page of the database. If the data was exported from a DB2 table, sub-table, or hierarchy, indexes are created. If this option operates on a hierarchy, and data was exported from DB2, a type hierarchy will also be created. This option can only be used with IXF files.

This parameter is not valid when you import to a nickname.

Note: If the data was exported from an MVS™ host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are more than 254, CREATE might fail because the rows are too long. See "Imported table re-creation" for a list of restrictions. In this case, the table should be created manually, and IMPORT with INSERT should be invoked, or, alternatively, the LOAD command should be used.

DEFAULT *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The schema specified through the DEFAULT clause identifies a schema to use for validation when the XML Data Specifier (XDS) of an imported XML document does not contain an SCH attribute identifying an XML Schema.

The DEFAULT clause takes precedence over the IGNORE and MAP clauses. If an XDS satisfies the DEFAULT clause, the IGNORE and MAP specifications will be ignored.

FROM *filename*

HIERARCHY
Specifies that hierarchical data is to be imported.

IGNORE *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The IGNORE clause specifies a list of one or more schemas to ignore if they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for an imported XML document, and the schema identified by the SCH attribute is included in the list of schemas to ignore, then no schema validation will occur for the imported XML document.

If a schema is specified in the IGNORE clause, it cannot also be present in the left side of a schema pair in the MAP clause.

The IGNORE clause applies only to the XDS. A schema that is mapped by the MAP clause will not be subsequently ignored if specified by the IGNORE clause.

IN *tablespace-name*
Identifies the table space in which the table will be created. The table space must exist, and must be a REGULAR table space. If no other table space is specified, all table parts are stored in this table space. If this clause is not specified, the table is created in a table space created by the authorization ID. If none is found, the table is placed into the default table space USERSPACE1. If USERSPACE1 has been dropped, table creation fails.

INDEX IN *tablespace-name*
Identifies the table space in which any indexes on the table will be created. This option is allowed only when the primary table space specified in the IN clause is a DMS table space. The specified table space must exist, and must be a REGULAR or LARGE DMS table space.

Note: Specifying which table space will contain an index can only be done when the table is created.

insert-column
Specifies the name of a column in the table or the view into which data is to be inserted.

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds rows of imported data to the target table, or updates existing rows (of the target table) with matching primary keys.

INTO *table-name*
Specifies the database table into which the data is to be imported. This table cannot be a system table, a declared temporary table or a summary table.

One can use an alias for INSERT, INSERT_UPDATE, or REPLACE, except in the case of an earlier server, when the fully qualified or the unqualified table name should be used. A qualified table name is in the form: *schema.tablename*. The *schema* is the user name under which the table was created.

LOBS FROM *lob-path*
The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

This parameter is not valid when you import to a nickname.

LONG IN *tablespace-name*
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, or distinct types with any of these as source types) will be stored. This option is allowed only if the primary table space specified in the IN clause is a DMS table space. The table space must exist, and must be a LARGE DMS table space.

MAP *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. Use the MAP clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each imported XML document. The MAP clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to

another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the MAP clause, it cannot also be specified in the IGNORE clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

METHOD

L Specifies the start and end column numbers from which to import data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1.

Note: This method can only be used with ASC files, and is the only valid option for that file type.

N Specifies the names of the columns in the data file to be imported. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the METHOD N list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, method N (F2, F1, F4, F3) is a valid request, while method N (F2, F1) is not valid.

Note: This method can only be used with IXF files.

P Specifies the field numbers of the input data fields to be imported.

Note: This method can only be used with IXF or DEL files, and is the only valid option for the DEL file type.

MODIFIED BY *filetype-mod*

Specifies file type modifier options. See "File type modifiers for the import utility" on page 76.

NOTIMEOUT

Specifies that the import utility will not time out while waiting for locks. This option supersedes the *locktimeout* database configuration parameter. Other applications are not affected.

NULL INDICATORS *null-indicator-list*

This option can only be used when the METHOD L parameter is specified. That is, the input file is an ASC file. The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the METHOD L parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column

specifies that the column data is not NULL, and that column data specified by the METHOD L option will be imported.

The NULL indicator character can be changed using the MODIFIED BY option, with the `nullindchar` file type modifier.

OF *filetype*

Specifies the format of the data in the input file:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony
- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2.

The WSF file type is not supported when you import to a nickname.

REPLACE

Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed. This option can only be used if the table exists. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

This option does not honor the CREATE TABLE statement's NOT LOGGED INITIALLY (NLI) clause or the ALTER TABLE statement's ACTIVE NOT LOGGED INITIALLY clause.

If an import with the REPLACE option is performed within the same transaction as a CREATE TABLE or ALTER TABLE statement where the NLI clause is invoked, the import will not honor the NLI clause. All inserts will be logged.

Workaround 1

Delete the contents of the table using the DELETE statement, then invoke the import with INSERT statement

Workaround 2

Drop the table and recreate it, then invoke the import with INSERT statement.

This limitation applies to DB2 Universal Database™ Version 7 and DB2 UDB Version 8

REPLACE_CREATE

Note: The REPLACE_CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

If the table exists, deletes all existing data from the table by truncating the data object, and inserts the imported data without changing the table definition or the index definitions.

If the table does not exist, creates the table and index definitions, as well as the row contents, in the code page of the database. See *Imported table re-creation* for a list of restrictions.

This option can only be used with IXF files. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

RESTARTCOUNT *n*
Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to SKIPCOUNT. RESTARTCOUNT and SKIPCOUNT are mutually exclusive.

ROWCOUNT *n*
Specifies the number *n* of physical records in the file to be imported (inserted or updated). Allows a user to import only *n* rows from a file, starting from the record determined by the SKIPCOUNT or RESTARTCOUNT options. If the SKIPCOUNT or RESTARTCOUNT options are not specified, the first *n* rows are imported. If SKIPCOUNT *m* or RESTARTCOUNT *m* is specified, rows *m*+1 to *m*+*n* are imported. When compound inserts are used, user specified ROWCOUNT *n* is rounded up to the first integer multiple of the compound count value.

SKIPCOUNT *n*
Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to RESTARTCOUNT. SKIPCOUNT and RESTARTCOUNT are mutually exclusive.

STARTING *sub-table-name*
A keyword for hierarchy only, requesting the default order, starting from *sub-table-name*. For PC/IXF files, the default order is the order stored in the input file. The default order is the only valid order for the PC/IXF file format.

sub-table-list
For typed tables with the INSERT or the INSERT_UPDATE option, a list of sub-table names is used to indicate the sub-tables into which data is to be imported.

traversal-order-list
For typed tables with the INSERT, INSERT_UPDATE, or the REPLACE option, a list of sub-table names is used to indicate the traversal order of the importing sub-tables in the hierarchy.

UNDER *sub-table-name*
Specifies a parent table for creating one or more sub-tables.

WARNINGCOUNT *n*
Stops the import operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail. If *n* is zero, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

XML FROM *xml-path*
Specifies one or more paths that contain the XML files.

XMLPARSE
Specifies how XML documents are parsed. If this option is not specified,

the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE
Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE
Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the XMLVALIDATE option is invoked with the USING XDS clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the DEFAULT clause.

The DEFAULT, IGNORE, and MAP clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the DEFAULT clause, it will not be ignored if also specified by the IGNORE clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the MAP clause, it will not be re-mapped if also specified in the second part of another MAP clause pair.

USING SCHEMA *schema-sqlid*
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the USING SCHEMALOCATION HINTS clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the XMLVALIDATE option below.

Usage notes

Be sure to complete all table operations and release all locks before starting an import operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

The import utility adds rows to the target table using the SQL INSERT statement. The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.

- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a REPLACE or a REPLACE_CREATE operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the REPLACE or the REPLACE_CREATE option to rerun the whole import operation, or use INSERT with the RESTARTCOUNT parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the INSERT or the INSERT_UPDATE option. They are, however, performed if the COMMITCOUNT parameter is not zero. If automatic COMMITs are not performed, a full log results in a ROLLBACK.

Offline import does not perform automatic COMMITs if any of the following conditions is true:

- the target is a view, not a table
- compound inserts are used
- buffered inserts are used

By default, online import performs automatic COMMITs to free both the active log space and the lock list. Automatic COMMITs are not performed only if a COMMITCOUNT value of zero is specified.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

You cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:

1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when recreating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using `SELECT *`.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60 KB), the message file returned to the user on the client might be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

Importing PC/IXF files to a remote database is much faster if the PC/IXF file is on a hard drive rather than on diskettes.

The database table or hierarchy must exist before data in the **ASC**, **DEL**, or **WSF** file formats can be imported; however, if the table does not already exist, IMPORT CREATE or IMPORT REPLACE_CREATE creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT CREATE can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand. The file copying step is not necessary if the source and the target databases are both accessible from the same client.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the FORCEIN option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the FORCEIN option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the FORCEIN option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 clients on the AIX operating system.

For table objects on an 8 KB page that are close to the limit of 1012 columns, import of PC/IXF data files might cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of **DEL** or **ASC** files. If PC/IXF files are being used to create a new table, an alternative is use db2look to dump the DDL statement that created the table, and then to issue that statement through the CLP.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (INSERT option) is supported. The RESTARTCOUNT parameter, but not the COMMITCOUNT parameter, is also supported.

When using the CREATE option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than CREATE with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file.

The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

Importing a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. Only the name of the first file must be specified in the IMPORT command. For example, `IMPORT FROM data.ixf OF IXF INSERT INTO TABLE1`. The file data.002, etc should be available in the same directory as `data.ixf`.

On the Windows operating system:

- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Security labels in their internal format might contain newline characters. If you import the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the `delprioritychar` file type modifier in the IMPORT command.

Federated considerations

When using the IMPORT command and the INSERT, UPDATE, or INSERT_UPDATE command parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you want to use when doing an import operation already exists. There are also several restrictions you should be aware of as shown in the IMPORT command parameters section.

Some data sources, such as ODBC, do not support importing into nicknames.

File type modifiers for the import utility

Table 14. Valid file type modifiers for the import utility: All file formats

Modifier	Description
compound=*x*	*x* is a number between 1 and 100 inclusive. Uses nonatomic compound SQL to insert the data, and *x* statements will be attempted each time. If this modifier is specified, and the transaction log is not sufficiently large, the import operation will fail. The transaction log must be large enough to accommodate either the number of rows specified by COMMITCOUNT, or the number of rows in the data file if COMMITCOUNT is not specified. It is therefore recommended that the COMMITCOUNT option be specified to avoid transaction log overflow. This modifier is incompatible with INSERT_UPDATE mode, hierarchical tables, and the following modifiers: `usedefaults`, `identitymissing`, `identityignore`, `generatedmissing`, and `generatedignore`.
generatedignore	This modifier informs the import utility that data for all generated columns is present in the data file but should be ignored. This results in all values for the generated columns being generated by the utility. This modifier cannot be used with the `generatedmissing` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated columns (not even NULLs), and will therefore generate a value for each row. This modifier cannot be used with the `generatedignore` modifier.
identityignore	This modifier informs the import utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `identitymissing` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with the `identityignore` modifier.
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. The LOBS FROM clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The LOBS FROM clause will implicitly activate the LOBSINFILE behavior. The LOBS FROM clause conveys to the IMPORT utility the list of paths to search for the LOB files while importing the data. To indicate a null LOB, enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
no_type_id	Valid only when importing into a single sub-table. Typical usage is to export data from a regular table, and then to invoke an import operation (using this modifier) to convert the data into a single sub-table.

Table 14. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
nodefaults	If a source column for a target table column is not explicitly specified, and the table column is not nullable, default values are not loaded. Without this option, if a source column for one of the target table columns is not explicitly specified, one of the following occurs: • If a default value can be specified for a column, the default value is loaded • If the column is nullable, and a default value cannot be specified for that column, a NULL is loaded • If the column is not nullable, and a default value cannot be specified, an error is returned, and the utility stops processing.
norowwarnings	Suppresses all warnings about rejected rows.
rowchangetimestampignore	This modifier informs the import utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMP being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `rowchangetimestampmissing` modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with the `rowchangetimestampignore` modifier.
seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. IMPORT converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W)) is returned. This modifier cannot be specified if the seclabelname modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned.
seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. IMPORT will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned. This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned. **Note:** If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.

Table 14. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. **Note:** For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 15. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data from this code page to the application code page during the import operation. The following rules apply: • For pure DBCS (graphic) mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. **Note:** 1. The `codepage` modifier cannot be used with the `lobsinfile` modifier. 2. If data expansion occurs when the code page is converted from the application code page to the database code page, the data might be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[2] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

Table 15. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.

timeformat="*x*"	*x* is the format of the time in the source file.[2] Valid time elements are:
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`TT - Meridian indicator (AM or PM)`
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 15. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[2] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements)`
	`UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microseond elements)`
	`UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microseond elements)`
	`UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microseond elements)`
	`UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microseond elements)`
	`U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microseond elements)`
	`TT - Meridian indicator (AM or PM)`
	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	The following example illustrates how to import data containing user defined date and time formats into a table called schedule:
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

Table 15. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being imported into graphic or double-byte character large object (DBCLOB) data fields is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic code page is associated with the character code page. IMPORT determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the application if the `codepage` modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The `usegraphiccodepage` modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The `usegraphiccodepage` modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the `codepage` file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute. **Note:** If the `xmlgraphic` modifier is specified with the IMPORT command, the XML document to be imported must be encoded in the UTF-16 code page. Otherwise, the XML document may be rejected with a parsing error, or it may be imported into the table with data corruption.

Table 16. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format

Modifier	Description
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Table 16. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format (continued)

Modifier	Description
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[3] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the null indicator character is specified to be the letter `N`, then `n` is also recognized as a null indicator.
reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.

striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.
	
	In the following example, `striptblanks` causes the import utility to truncate trailing blank spaces:
	`db2 import from myfile.asc of asc`
	`  modified by striptblanks`
	`  method l (1 10, 12 15) messages msgs.txt`
	`  insert into staff`
	
	This option cannot be specified together with `striptnulls`. These are mutually exclusive options. This option replaces the obsolete `t` option, which is supported for earlier compatibility only.

striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. This option replaces the obsolete `padwithzero` option, which is supported for earlier compatibility only.

Table 17. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[34] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	
	The single quotation mark (′) can also be specified as a character string delimiter. In the following example, `chardel''` causes the import utility to interpret any single quotation mark (′) it encounters as a character string delimiter:
	`db2 "import from myfile.del of del`
	`  modified by chardel''`
	`  method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[34]
	
	In the following example, `coldel;` causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 import from myfile.del of del`
	`  modified by coldel;`
	`  messages msgs.txt insert into staff`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.

Table 17. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[34]
	
	In the following example, `decpt;` causes the import utility to interpret any semicolon (;) it encounters as a decimal point:
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`decpt; messages msgs.txt insert into staff"`

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 import ... modified by delprioritychar ...`
	
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
nochardel	The import utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx`, `delprioritychar` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 18. Valid file type modifiers for the import utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to import each row.
indexixf	Directs the utility to drop all indexes currently defined on the existing table, and to create new ones from the index definitions in the PC/IXF file. This option can only be used when the contents of a table are being replaced. It cannot be used with a view, or when a *insert-column* is specified.

Table 18. Valid file type modifiers for the import utility: IXF file format (continued)

Modifier	Description
indexschema=*schema*	Uses the specified *schema* for the index name during index creation. If *schema* is not specified (but the keyword `indexschema` *is* specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
forcecreate	Specifies that the table should be created with possible missing or limited information after returning SQL3311N during an import operation.

Table 19. IMPORT behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	IMPORT behavior
Absent	Absent	All data in the file is assumed to be in the application code page.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the application code page. Graphic data is assumed to be in the code page of the application graphic data. If the application code page is single-byte, then all data is assumed to be in the application code page. **Warning:** If the application code page is single-byte, graphic data will be corrupted when imported into the database, even if the database contains graphic columns.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.

Note:

1. The import utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the import operation fails, and an error code is returned.
2. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end

positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

```
"M" (could be a month, or a minute)
"M:M" (Which is which?)
"M:YYYY:M" (Both are interpreted as month.)
"S:M:YYYY" (adjacent to both a time value and a date value)
```

In ambiguous cases, the utility will report an error message, and the operation will fail.

Following are some unambiguous time stamp formats:

```
"M:YYYY" (Month)
"S:M" (Minute)
"M:YYYY:S:M" (Month....Minute)
"M:H:YYYY:M:D" (Minute....Month)
```

Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).

3. Character values provided for the chardel, coldel, or decpt file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```

4. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
5. The following file type modifiers are not allowed when importing into a nickname:
   - `indexixf`
   - `indexschema`
   - `dldelfiletype`
   - `nodefaults`
   - `usedefaults`
   - `no_type_idfiletype`
   - `generatedignore`
   - `generatedmissing`
   - `identityignore`
   - `identitymissing`
   - `lobsinfile`
6. The **WSF** file format is not supported for XML columns.
7. The CREATE mode is not supported for XML columns.
8. All XML data must reside in XML files that are separate from the main data file. An XML Data Specifier (XDS) (or a NULL value) must exist for each XML column in the main data file.

9. XML documents are assumed to be in Unicode format or to contain a declaration tag that includes an encoding attribute, unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. Rows containing documents that are not well-formed will be rejected.
11. If the XMLVALIDATE option is specified, documents that successfully validate against their matching schema will be annotated with the schema information as they are inserted. Rows containing documents that fail to validate against their matching schema will be rejected. To successfully perform the validation, the privileges held by the user invoking the import must include at least one of the following:
    - SYSADM or DBADM authority
    - USAGE privilege on the XML schema to be used in the validation
12. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the rowchangetimestampmissing file type modifier *must be* specified in the import command if data for the column is not present in the data to be imported and there is no explicit column list present.

IMPORT command using the ADMIN_CMD procedure

Inserts data from an external file with a supported file format into a table, hierarchy, view or nickname. LOAD is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Quick link to "File type modifiers for the import utility" on page 99.

Authorization

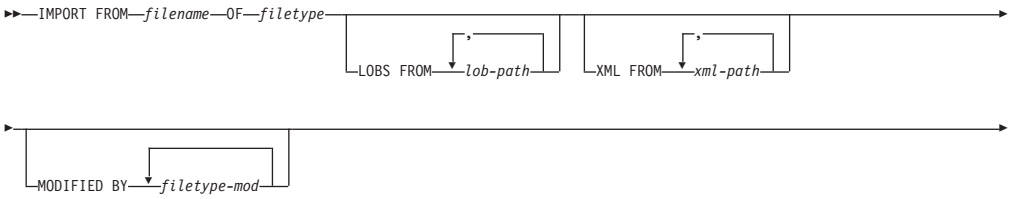
- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*

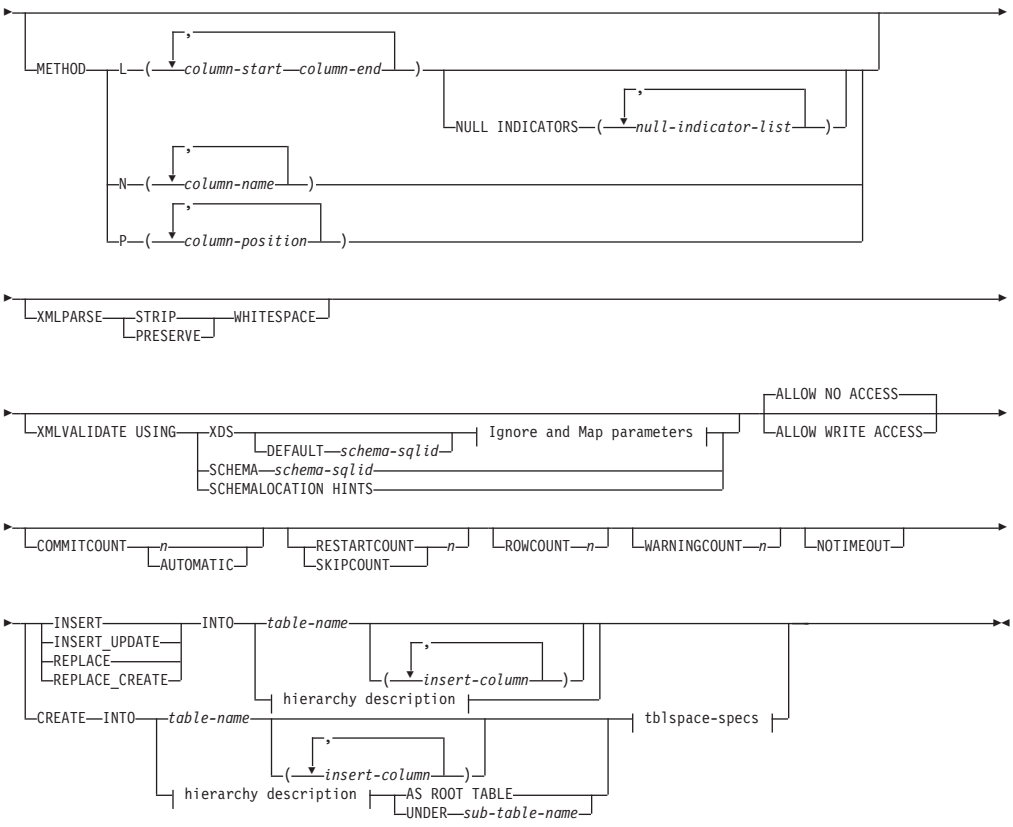
  - CREATETAB authority on the database and USE privilege on the table space, as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- IMPORT to a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CREATETAB authority on the database and USE privilege on the table space and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the REPLACE_CREATE option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the REPLACE option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on every sub-table in the hierarchy
- To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the import fails and an error (SQLSTATE 42512) is returned.
- To import data into a table that has protected rows, the session authorization ID must hold LBAC credentials that meet these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access

  The label on the row to insert, the user's LBAC credentials, the security policy definition, and the LBAC rules determine the label on the row.
- If the REPLACE or REPLACE_CREATE option is specified, the session authorization ID must have the authority to drop the table.

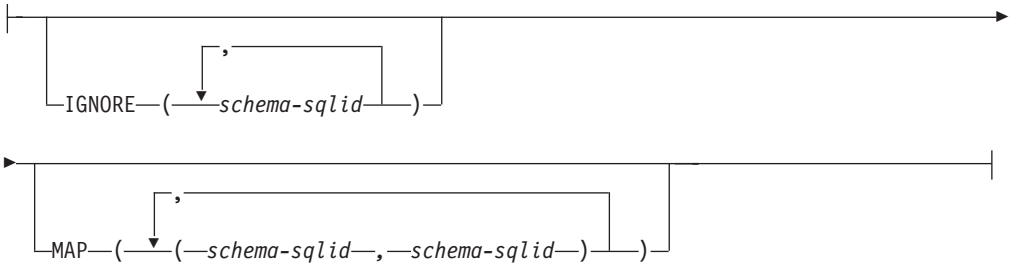
Required connection

Command syntax



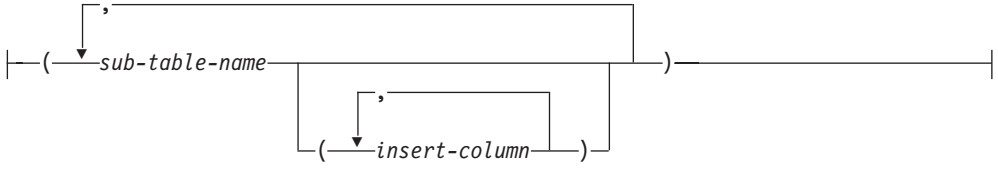


Ignore and Map parameters:



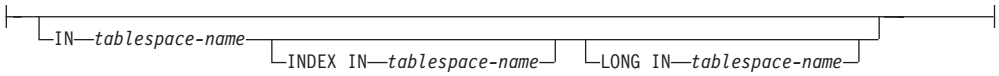
hierarchy description:

sub-table-list:



traversal-order-list:

tblspace-specs:



Command parameters

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal order.

ALLOW NO ACCESS
Runs import in the offline mode. An exclusive (X) lock on the target table is acquired before any rows are inserted. This prevents concurrent applications from accessing table data. This is the default import behavior.

ALLOW WRITE ACCESS
Runs import in the online mode. An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the REPLACE, CREATE, or REPLACE_CREATE import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the COMMITCOUNT option was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required when you import to a nickname and COMMITCOUNT must be specified with a valid number (AUTOMATIC is not considered a valid option).

AS ROOT TABLE
Creates one or more sub-tables as a stand-alone table hierarchy.

COMMITCOUNT *n* **| AUTOMATIC**
Performs a COMMIT after every *n* records are imported. When a number *n* is specified, import performs a COMMIT after every *n* records are imported. When compound inserts are used, a user-specified commit frequency of *n* is rounded up to the first integer multiple of the compound count value. When AUTOMATIC is specified, import internally determines when a commit needs to be performed. The utility will commit for either one of two reasons:

- to avoid running out of active log space
- to avoid lock escalation from row level to table level

If the ALLOW WRITE ACCESS option is specified, and the COMMITCOUNT option is not specified, the import utility will perform commits as if COMMITCOUNT AUTOMATIC had been specified.

If the IMPORT command encounters an SQL0964C (Transaction Log Full) while inserting or updating a record and the COMMITCOUNT *n* is specified, IMPORT will attempt to resolve the issue by performing an

unconditional commit and then reattempt to insert or update the record. If this does not help resolve the log full condition (which would be the case when the log full is attributed to other activity on the database), then the IMPORT command will fail as expected, however the number of rows committed may not be a multiple of the COMMITCOUNT *n* value. The RESTARTCOUNT or SKIPCOUNT option can be used to avoid processing those row already committed.

CREATE

Note: The CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

Creates the table definition and row contents in the code page of the database. If the data was exported from a DB2 table, sub-table, or hierarchy, indexes are created. If this option operates on a hierarchy, and data was exported from DB2, a type hierarchy will also be created. This option can only be used with IXF files.

This parameter is not valid when you import to a nickname.

Note: If the data was exported from an MVS host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are more than 254, CREATE might fail because the rows are too long. See "Imported table re-creation" for a list of restrictions. In this case, the table should be created manually, and IMPORT with INSERT should be invoked, or, alternatively, the LOAD command should be used.

DEFAULT *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The schema specified through the DEFAULT clause identifies a schema to use for validation when the XML Data Specifier (XDS) of an imported XML document does not contain an SCH attribute identifying an XML Schema.

The DEFAULT clause takes precedence over the IGNORE and MAP clauses. If an XDS satisfies the DEFAULT clause, the IGNORE and MAP specifications will be ignored.

FROM *filename*

HIERARCHY
Specifies that hierarchical data is to be imported.

IGNORE *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The IGNORE clause specifies a list of one or more schemas to ignore if they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for an imported XML document, and the schema identified by the SCH attribute is included in the list of schemas to ignore, then no schema validation will occur for the imported XML document.

If a schema is specified in the IGNORE clause, it cannot also be present in the left side of a schema pair in the MAP clause.

The IGNORE clause applies only to the XDS. A schema that is mapped by the MAP clause will not be subsequently ignored if specified by the IGNORE clause.

IN *tablespace-name*
Identifies the table space in which the table will be created. The table space must exist, and must be a REGULAR table space. If no other table space is specified, all table parts are stored in this table space. If this clause is not specified, the table is created in a table space created by the authorization ID. If none is found, the table is placed into the default table space USERSPACE1. If USERSPACE1 has been dropped, table creation fails.

INDEX IN *tablespace-name*
Identifies the table space in which any indexes on the table will be created. This option is allowed only when the primary table space specified in the IN clause is a DMS table space. The specified table space must exist, and must be a REGULAR or LARGE DMS table space.

Note: Specifying which table space will contain an index can only be done when the table is created.

insert-column
Specifies the name of a column in the table or the view into which data is to be inserted.

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds rows of imported data to the target table, or updates existing rows (of the target table) with matching primary keys.

INTO *table-name*
Specifies the database table into which the data is to be imported. This table cannot be a system table, a declared temporary table or a summary table.

One can use an alias for INSERT, INSERT_UPDATE, or REPLACE, except in the case of an earlier server, when the fully qualified or the unqualified table name should be used. A qualified table name is in the form: *schema.tablename*. The *schema* is the user name under which the table was created.

LOBS FROM *lob-path*
The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

This parameter is not valid when you import to a nickname.

LONG IN *tablespace-name*
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, or distinct types with any of these as source types) will be stored. This option is allowed only if the primary table space specified in the IN clause is a DMS table space. The table space must exist, and must be a LARGE DMS table space.

MAP *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. Use the MAP clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each imported XML document. The MAP clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to

another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the MAP clause, it cannot also be specified in the IGNORE clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

METHOD

L Specifies the start and end column numbers from which to import data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1.

Note: This method can only be used with ASC files, and is the only valid option for that file type.

N Specifies the names of the columns in the data file to be imported. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the METHOD N list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, method N (F2, F1, F4, F3) is a valid request, while method N (F2, F1) is not valid.

Note: This method can only be used with IXF files.

P Specifies the field numbers of the input data fields to be imported.

Note: This method can only be used with IXF or DEL files, and is the only valid option for the DEL file type.

MODIFIED BY *filetype-mod*

Specifies file type modifier options. See "File type modifiers for the import utility" on page 99.

NOTIMEOUT

Specifies that the import utility will not time out while waiting for locks. This option supersedes the *locktimeout* database configuration parameter. Other applications are not affected.

NULL INDICATORS *null-indicator-list*

This option can only be used when the METHOD L parameter is specified. That is, the input file is an ASC file. The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the METHOD L parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column

specifies that the column data is not NULL, and that column data specified by the METHOD L option will be imported.

The NULL indicator character can be changed using the MODIFIED BY option, with the `nullindchar` file type modifier.

OF *filetype*

Specifies the format of the data in the input file:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony
- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2.

The WSF file type is not supported when you import to a nickname.

REPLACE

Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed. This option can only be used if the table exists. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

This option does not honor the CREATE TABLE statement's NOT LOGGED INITIALLY (NLI) clause or the ALTER TABLE statement's ACTIVE NOT LOGGED INITIALLY clause.

If an import with the REPLACE option is performed within the same transaction as a CREATE TABLE or ALTER TABLE statement where the NLI clause is invoked, the import will not honor the NLI clause. All inserts will be logged.

Workaround 1

Delete the contents of the table using the DELETE statement, then invoke the import with INSERT statement

Workaround 2

Drop the table and recreate it, then invoke the import with INSERT statement.

This limitation applies to DB2 Universal Database Version 7 and DB2 UDB Version 8

REPLACE_CREATE

Note: The REPLACE_CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

If the table exists, deletes all existing data from the table by truncating the data object, and inserts the imported data without changing the table definition or the index definitions.

If the table does not exist, creates the table and index definitions, as well as the row contents, in the code page of the database. See *Imported table re-creation* for a list of restrictions.

This option can only be used with IXF files. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

RESTARTCOUNT *n*
Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to SKIPCOUNT. RESTARTCOUNT and SKIPCOUNT are mutually exclusive.

ROWCOUNT *n*
Specifies the number *n* of physical records in the file to be imported (inserted or updated). Allows a user to import only *n* rows from a file, starting from the record determined by the SKIPCOUNT or RESTARTCOUNT options. If the SKIPCOUNT or RESTARTCOUNT options are not specified, the first *n* rows are imported. If SKIPCOUNT *m* or RESTARTCOUNT *m* is specified, rows *m*+1 to *m*+*n* are imported. When compound inserts are used, user specified ROWCOUNT *n* is rounded up to the first integer multiple of the compound count value.

SKIPCOUNT *n*
Specifies that an import operation is to be started at record *n*+1. The first *n* records are skipped. This option is functionally equivalent to RESTARTCOUNT. SKIPCOUNT and RESTARTCOUNT are mutually exclusive.

STARTING *sub-table-name*
A keyword for hierarchy only, requesting the default order, starting from *sub-table-name*. For PC/IXF files, the default order is the order stored in the input file. The default order is the only valid order for the PC/IXF file format.

sub-table-list
For typed tables with the INSERT or the INSERT_UPDATE option, a list of sub-table names is used to indicate the sub-tables into which data is to be imported.

traversal-order-list
For typed tables with the INSERT, INSERT_UPDATE, or the REPLACE option, a list of sub-table names is used to indicate the traversal order of the importing sub-tables in the hierarchy.

UNDER *sub-table-name*
Specifies a parent table for creating one or more sub-tables.

WARNINGCOUNT *n*
Stops the import operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail. If *n* is zero, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

XML FROM *xml-path*
Specifies one or more paths that contain the XML files.

XMLPARSE
Specifies how XML documents are parsed. If this option is not specified,

the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE
Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE
Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the XMLVALIDATE option is invoked with the USING XDS clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the DEFAULT clause.

The DEFAULT, IGNORE, and MAP clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the DEFAULT clause, it will not be ignored if also specified by the IGNORE clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the MAP clause, it will not be re-mapped if also specified in the second part of another MAP clause pair.

USING SCHEMA *schema-sqlid*
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the USING SCHEMALOCATION HINTS clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the XMLVALIDATE option below.

Usage notes

Be sure to complete all table operations and release all locks before starting an import operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

The import utility adds rows to the target table using the SQL INSERT statement. The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.

- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a REPLACE or a REPLACE_CREATE operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the REPLACE or the REPLACE_CREATE option to rerun the whole import operation, or use INSERT with the RESTARTCOUNT parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the INSERT or the INSERT_UPDATE option. They are, however, performed if the COMMITCOUNT parameter is not zero. If automatic COMMITs are not performed, a full log results in a ROLLBACK.

Offline import does not perform automatic COMMITs if any of the following conditions is true:

- the target is a view, not a table
- compound inserts are used
- buffered inserts are used

By default, online import performs automatic COMMITs to free both the active log space and the lock list. Automatic COMMITs are not performed only if a COMMITCOUNT value of zero is specified.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

You cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:

1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when recreating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using `SELECT *`.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60 KB), the message file returned to the user on the client might be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

Importing PC/IXF files to a remote database is much faster if the PC/IXF file is on a hard drive rather than on diskettes.

The database table or hierarchy must exist before data in the **ASC**, **DEL**, or **WSF** file formats can be imported; however, if the table does not already exist, IMPORT CREATE or IMPORT REPLACE_CREATE creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT CREATE can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand. The file copying step is not necessary if the source and the target databases are both accessible from the same client.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the FORCEIN option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the FORCEIN option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the FORCEIN option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 clients on the AIX operating system.

For table objects on an 8 KB page that are close to the limit of 1012 columns, import of PC/IXF data files might cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of **DEL** or **ASC** files. If PC/IXF files are being used to create a new table, an alternative is use db2look to dump the DDL statement that created the table, and then to issue that statement through the CLP.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (INSERT option) is supported. The RESTARTCOUNT parameter, but not the COMMITCOUNT parameter, is also supported.

When using the CREATE option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than CREATE with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file.

The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

Importing a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. Only the name of the first file must be specified in the IMPORT command. For example, `IMPORT FROM data.ixf OF IXF INSERT INTO TABLE1`. The file data.002, etc should be available in the same directory as `data.ixf`.

On the Windows operating system:

- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Security labels in their internal format might contain newline characters. If you import the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the `delprioritychar` file type modifier in the IMPORT command.

Federated considerations

When using the IMPORT command and the INSERT, UPDATE, or INSERT_UPDATE command parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you want to use when doing an import operation already exists. There are also several restrictions you should be aware of as shown in the IMPORT command parameters section.

Some data sources, such as ODBC, do not support importing into nicknames.

File type modifiers for the import utility

Table 20. Valid file type modifiers for the import utility: All file formats

Modifier	Description
compound=*x*	*x* is a number between 1 and 100 inclusive. Uses nonatomic compound SQL to insert the data, and *x* statements will be attempted each time. If this modifier is specified, and the transaction log is not sufficiently large, the import operation will fail. The transaction log must be large enough to accommodate either the number of rows specified by COMMITCOUNT, or the number of rows in the data file if COMMITCOUNT is not specified. It is therefore recommended that the COMMITCOUNT option be specified to avoid transaction log overflow. This modifier is incompatible with INSERT_UPDATE mode, hierarchical tables, and the following modifiers: `usedefaults`, `identitymissing`, `identityignore`, `generatedmissing`, and `generatedignore`.
generatedignore	This modifier informs the import utility that data for all generated columns is present in the data file but should be ignored. This results in all values for the generated columns being generated by the utility. This modifier cannot be used with the `generatedmissing` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated columns (not even NULLs), and will therefore generate a value for each row. This modifier cannot be used with the `generatedignore` modifier.
identityignore	This modifier informs the import utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `identitymissing` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with the `identityignore` modifier.
lobsinfile	*lob-path* specifies the path to the files containing LOB data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. The LOBS FROM clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The LOBS FROM clause will implicitly activate the LOBSINFILE behavior. The LOBS FROM clause conveys to the IMPORT utility the list of paths to search for the LOB files while importing the data. To indicate a null LOB, enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
no_type_id	Valid only when importing into a single sub-table. Typical usage is to export data from a regular table, and then to invoke an import operation (using this modifier) to convert the data into a single sub-table.

Table 20. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
nodefaults	If a source column for a target table column is not explicitly specified, and the table column is not nullable, default values are not loaded. Without this option, if a source column for one of the target table columns is not explicitly specified, one of the following occurs: • If a default value can be specified for a column, the default value is loaded • If the column is nullable, and a default value cannot be specified for that column, a NULL is loaded • If the column is not nullable, and a default value cannot be specified, an error is returned, and the utility stops processing.
norowwarnings	Suppresses all warnings about rejected rows.
rowchangetimestampignore	This modifier informs the import utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMP being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `rowchangetimestampmissing` modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with the `rowchangetimestampignore` modifier.
seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. IMPORT converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W)) is returned. This modifier cannot be specified if the seclabelname modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned.
seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. IMPORT will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned. This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned. **Note:** If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.

Table 20. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. **Note:** For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 21. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data from this code page to the application code page during the import operation. The following rules apply: • For pure DBCS (graphic) mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. **Note:** 1. The `codepage` modifier cannot be used with the `lobsinfile` modifier. 2. If data expansion occurs when the code page is converted from the application code page to the database code page, the data might be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[2] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

Table 21. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.

timeformat="*x*"	*x* is the format of the time in the source file.[2] Valid time elements are:
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`TT - Meridian indicator (AM or PM)`
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 21. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[2] Valid time stamp elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)`
	`MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements)`
	`UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microseond elements)`
	`UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microseond elements)`
	`UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microseond elements)`
	`UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microseond elements)`
	`U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microseond elements)`
	`TT - Meridian indicator (AM or PM)`
	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	The following example illustrates how to import data containing user defined date and time formats into a table called schedule:
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

Table 21. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being imported into graphic or double-byte character large object (DBCLOB) data fields is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic code page is associated with the character code page. IMPORT determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the application if the `codepage` modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The `usegraphiccodepage` modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The `usegraphiccodepage` modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the `codepage` file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute. **Note:** If the `xmlgraphic` modifier is specified with the IMPORT command, the XML document to be imported must be encoded in the UTF-16 code page. Otherwise, the XML document may be rejected with a parsing error, or it may be imported into the table with data corruption.

Table 22. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format

Modifier	Description
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Table 22. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format (continued)

Modifier	Description
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[3] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the null indicator character is specified to be the letter `N`, then `n` is also recognized as a null indicator.
reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.

striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.
	
	In the following example, `striptblanks` causes the import utility to truncate trailing blank spaces:
	`db2 import from myfile.asc of asc`
	`modified by striptblanks`
	`method l (1 10, 12 15) messages msgs.txt`
	`insert into staff`
	
	This option cannot be specified together with `striptnulls`. These are mutually exclusive options. This option replaces the obsolete `t` option, which is supported for earlier compatibility only.

striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. This option replaces the obsolete `padwithzero` option, which is supported for earlier compatibility only.

Table 23. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[34] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	
	The single quotation mark (') can also be specified as a character string delimiter. In the following example, `chardel''` causes the import utility to interpret any single quotation mark (') it encounters as a character string delimiter:
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[34]
	
	In the following example, `coldel;` causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 import from myfile.del of del`
	`modified by coldel;`
	`messages msgs.txt insert into staff`

decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.

Table 23. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[34]
	
	In the following example, `decpt;` causes the import utility to interpret any semicolon (;) it encounters as a decimal point:
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`decpt; messages msgs.txt insert into staff"`

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 import ... modified by delprioritychar ...`
	
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
nochardel	The import utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx`, `delprioritychar` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 24. Valid file type modifiers for the import utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to import each row.
indexixf	Directs the utility to drop all indexes currently defined on the existing table, and to create new ones from the index definitions in the PC/IXF file. This option can only be used when the contents of a table are being replaced. It cannot be used with a view, or when a *insert-column* is specified.

Table 24. Valid file type modifiers for the import utility: IXF file format (continued)

Modifier	Description
indexschema=*schema*	Uses the specified *schema* for the index name during index creation. If *schema* is not specified (but the keyword `indexschema` *is* specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
forcecreate	Specifies that the table should be created with possible missing or limited information after returning SQL3311N during an import operation.

Table 25. IMPORT behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	IMPORT behavior
Absent	Absent	All data in the file is assumed to be in the application code page.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the application code page. Graphic data is assumed to be in the code page of the application graphic data. If the application code page is single-byte, then all data is assumed to be in the application code page. **Warning:** If the application code page is single-byte, graphic data will be corrupted when imported into the database, even if the database contains graphic columns.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.

Note:

1. The import utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the import operation fails, and an error code is returned.
2. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end

positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

```
"M" (could be a month, or a minute)
"M:M" (Which is which?)
"M:YYYY:M" (Both are interpreted as month.)
"S:M:YYYY" (adjacent to both a time value and a date value)
```

In ambiguous cases, the utility will report an error message, and the operation will fail.

Following are some unambiguous time stamp formats:

```
"M:YYYY" (Month)
"S:M" (Minute)
"M:YYYY:S:M" (Month....Minute)
"M:H:YYYY:M:D" (Minute....Month)
```

Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).

3. Character values provided for the chardel, coldel, or decpt file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```

4. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.
5. The following file type modifiers are not allowed when importing into a nickname:
   - `indexixf`
   - `indexschema`
   - `dldelfiletype`
   - `nodefaults`
   - `usedefaults`
   - `no_type_idfiletype`
   - `generatedignore`
   - `generatedmissing`
   - `identityignore`
   - `identitymissing`
   - `lobsinfile`
6. The **WSF** file format is not supported for XML columns.
7. The CREATE mode is not supported for XML columns.
8. All XML data must reside in XML files that are separate from the main data file. An XML Data Specifier (XDS) (or a NULL value) must exist for each XML column in the main data file.

9. XML documents are assumed to be in Unicode format or to contain a declaration tag that includes an encoding attribute, unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. Rows containing documents that are not well-formed will be rejected.
11. If the XMLVALIDATE option is specified, documents that successfully validate against their matching schema will be annotated with the schema information as they are inserted. Rows containing documents that fail to validate against their matching schema will be rejected. To successfully perform the validation, the privileges held by the user invoking the import must include at least one of the following:
    - SYSADM or DBADM authority
    - USAGE privilege on the XML schema to be used in the validation
12. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the rowchangetimestampmissing file type modifier *must be* specified in the import command if data for the column is not present in the data to be imported and there is no explicit column list present.

db2Import - Import data into a table, hierarchy, nickname or view

Inserts data from an external file with a supported file format into a table, hierarchy, nickname or view. The load utility is faster than this function. The load utility, however, does not support loading data at the hierarchy level or loading into a nickname.

Authorization

- IMPORT using the INSERT option requires one of the following:
  - sysadm
  - dbadm
  - CONTROL privilege on each participating table, view or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
  - sysadm
  - dbadm
  - CONTROL privilege on the table, view or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
  - sysadm
  - dbadm
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:
  - sysadm
  - dbadm

  - CREATETAB authority on the database and USE privilege on the table space, as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- IMPORT to a table or a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  - sysadm
  - dbadm
  - CREATETAB authority on the database, and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the REPLACE_CREATE option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the REPLACE option requires one of the following:
  - sysadm
  - dbadm
  - CONTROL privilege on every sub-table in the hierarchy

Required connection

Database. If implicit connect is enabled, a connection to the default database is established.

API include file

`db2ApiDf.h`

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Import (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ImportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ImportIn *piImportInfoIn;
   struct db2ImportOut *poImportInfoOut;
   db2int32 *piNullIndicators;
   struct sqllob *piLongActionString;
} db2ImportStruct;

typedef SQL_STRUCTURE db2ImportIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
```

```
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2ImportIn;

typedef SQL_STRUCTURE db2ImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2ImportOut;

typedef SQL_STRUCTURE db2DMUXmlMapSchema
{
   struct db2Char                       iMapFromSchema;
   struct db2Char                       iMapToSchema;
} db2DMUXmlMapSchema;

typedef SQL_STRUCTURE db2DMUXmlValidateXds
{
   struct db2Char *piDefaultSchema;
   db2Uint32 iNumIgnoreSchemas;
   struct db2Char *piIgnoreSchemas;
   db2Uint32 iNumMapSchemas;
   struct db2DMUXmlMapSchema *piMapSchemas;
} db2DMUXmlValidateXds;

typedef SQL_STRUCTURE db2DMUXmlValidateSchema
{
   struct db2Char *piSchema;
} db2DMUXmlValidateSchema;

typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16 iUsing;
   struct db2DMUXmlValidateXds *piXdsArgs;
   struct db2DMUXmlValidateSchema *piSchemaArgs;
} db2DMUXmlValidate;

SQL_API_RC SQL_API_FN
  db2gImport (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gImportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2gImportIn *piImportInfoIn;
   struct dbg2ImportOut *poImportInfoOut;
   db2int32 *piNullIndicators;
   db2Uint16 iDataFileNameLen;
   db2Uint16 iFileTypeLen;
```

```
   db2Uint16 iMsgFileNameLen;
   struct sqllob *piLongActionString;
} db2gImportStruct;

typedef SQL_STRUCTURE db2gImportIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2gImportIn;

typedef SQL_STRUCTURE db2gImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2gImportOut;
```

db2Import API parameters

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter pParmStruct.

pParmStruct
Input/Output. A pointer to the db2ImportStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ImportStruct data structure parameters

piDataFileName
Input. A string containing the path and the name of the external input file from which the data is to be imported.

piLobPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files can be found. This parameter is not valid when you import to a nickname.

piDataDescriptor
Input. Pointer to an sqldcol structure containing information about the columns being selected for import from the external file. The value of the dcolmeth field determines how the remainder of the information provided in this parameter is interpreted by the import utility. Valid values for this parameter are:

SQL_METH_N
Names. Selection of columns from the external input file is by column name.

SQL_METH_P
Positions. Selection of columns from the external input file is by column position.

SQL_METH_L
Locations. Selection of columns from the external input file is by column location. The database manager rejects an import call with a location pair that is invalid because of any one of the following conditions:

- Either the beginning or the ending location is not in the range from 1 to the largest signed 2-byte integer.
- The ending location is smaller than the beginning location.
- The input column width defined by the location pair is not compatible with the type and the length of the target column.

A location pair with both locations equal to zero indicates that a nullable column is to be filled with NULLs.

SQL_METH_D
Default. If piDataDescriptor is NULL, or is set to SQL_METH_D, default selection of columns from the external input file is done. In this case, the number of columns and the column specification array are both ignored. For DEL, IXF, or WSF files, the first n columns of data in the external input file are taken in their natural order, where n is the number of database columns into which the data is to be imported.

piActionString
Deprecated. Replaced by piLongActionString.

piLongActionString
Input. Pointer to an *sqllob* structure containing a 4-byte long field, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
{INSERT | INSERT_UPDATE | REPLACE | CREATE | REPLACE_CREATE}
INTO {tname[(tcolumn-list)] |
[{ALL TABLES | (tname[(tcolumn-list)][, tname[(tcolumn-list)]])}]
[IN] HIERARCHY {STARTING tname | (tname[, tname])}
[UNDER sub-table-name | AS ROOT TABLE]}
```

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds the imported rows if their primary key values are not in the table, and uses them for update if their primary key values are found. This option is only valid if the target table has a primary key, and the specified (or implied) list of target columns being imported includes all columns for the primary key. This option cannot be applied to views.

REPLACE
Deletes all existing data from the table by truncating the table object, and inserts the imported data. The table definition and the index definitions are not changed. (Indexes are deleted and replaced if indexixf is in FileTypeMod, and FileType is SQL_IXF.) If the table is not already defined, an error is returned.

Note: If an error occurs after the existing data is deleted, that data is lost.
This parameter is not valid when you import to a nickname.

CREATE

Note: The CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

Creates the table definition and the row contents using the information in the specified PC/IXF file, if the specified table is not defined. If the file was previously exported by DB2, indexes are also created. If the specified table is already defined, an error is returned. This option is valid for the PC/IXF file format only. This parameter is not valid when you import to a nickname.

REPLACE_CREATE

Note: The REPLACE_CREATE parameter is deprecated and may be removed in a future release. For additional details, see "IMPORT command options CREATE and REPLACE_CREATE are deprecated".

Replaces the table contents using the PC/IXF row information in the PC/IXF file, if the specified table is defined. If the table is not already defined, the table definition and row contents are created using the information in the specified PC/IXF file. If the PC/IXF file was previously exported by DB2, indexes are also created. This option is valid for the PC/IXF file format only.

Note: If an error occurs after the existing data is deleted, that data is lost.
This parameter is not valid when you import to a nickname.

tname The name of the table, typed table, view, or object view into which the data is to be inserted. An alias for REPLACE, INSERT_UPDATE, or INSERT can be specified, except in the case of an earlier server, when a qualified or unqualified name should be specified. If it is a view, it cannot be a read-only view.

tcolumn-list
A list of table or view column names into which the data is to be inserted. The column names must be separated by commas. If column names are not specified, column names as defined in the CREATE TABLE or the ALTER TABLE statement are used. If no column list is specified for typed tables, data is inserted into all columns within each sub-table.

sub-table-name
Specifies a parent table when creating one or more sub-tables under the CREATE option.

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal-order-list.

HIERARCHY
Specifies that hierarchical data is to be imported.

STARTING
Keyword for hierarchy only. Specifies that the default order, starting from a given sub-table name, is to be used.

UNDER
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created under a given sub-table.

AS ROOT TABLE
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created as a stand-alone hierarchy.

The tname and the tcolumn-list parameters correspond to the tablename and the colname lists of SQL INSERT statements, and have the same restrictions.

The columns in tcolumn-list and the external columns (either specified or implied) are matched according to their position in the list or the structure (data from the first column specified in the sqldcol structure is inserted into the table or view field corresponding to the first element of the tcolumn-list).

If unequal numbers of columns are specified, the number of columns actually processed is the lesser of the two numbers. This could result in an error (because there are no values to place in some non-nullable table fields) or an informational message (because some external file columns are ignored).

This parameter is not valid when you import to a nickname.

piFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integration Exchange Format, the preferred method for exporting data from a table so that it can be imported later into the same table or into another database manager table.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs. The WSF file type is not supported when you import to a nickname.

piFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link "File type modifiers for the import utility".

piMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, it is appended to. If it does not exist, a file is created.

iCallerAction
Input. An action requested by the caller. Valid values are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API. If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested import operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

piImportInfoIn
Input. Pointer to the db2ImportIn structure.

poImportInfoOut
Output. Pointer to the db2ImportOut structure.

piNullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. The number of elements in this array must match the number of columns in the input file; there is a one-to-one ordered correspondence between the elements of this array and the columns being imported from the data file. Therefore, the number of elements must equal the dcolnum field of the piDataDescriptor parameter. Each element of the array contains a number identifying a column in the data file that is to be used as a null indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified column in the data file must contain a Y or an N. A Y indicates that the table column data is NULL, and N indicates that the table column data is not NULL.

piXmlPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files can be found.

db2ImportIn data structure parameters

iRowcount
Input. The number of physical records to be loaded. Allows a user to load only the first iRowcount rows in a file. If iRowcount is 0, import will attempt to process all the rows from the file.

iRestartcount
Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iSkipcount parameter. iRestartcount and iSkipcount parameters are mutually exclusive.

iSkipcount
Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iRestartcount.

piCommitcount
Input. The number of records to import before committing them to the database. A commit is performed whenever piCommitcount records are imported. A NULL value specifies the default commit count value, which is zero for offline import and AUTOMATIC for online import. Commitcount AUTOMATIC is specified by passing in the value DB2IMPORT_COMMIT_AUTO.

iWarningcount
Input. Stops the import operation after iWarningcount warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail.

If iWarningcount is 0, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

iNoTimeout
Input. Specifies that the import utility will not time out while waiting for locks. This option supersedes the locktimeout database configuration parameter. Other applications are not affected. Valid values are:

DB2IMPORT_LOCKTIMEOUT
Indicates that the value of the locktimeout configuration parameter is respected.

DB2IMPORT_NO_LOCKTIMEOUT
Indicates there is no timeout.

iAccessLevel
Input. Specifies the access level. Valid values are:

- SQLU_ALLOW_NO_ACCESS
Specifies that the import utility locks the table exclusively.

- SQLU_ALLOW_WRITE_ACCESS
Specifies that the data in the table should still be accessible to readers and writers while the import is in progress.

An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the REPLACE, CREATE, or REPLACE_CREATE import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid

running out of active log space. These commits will be performed even if the piCommitCount parameter was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required when you import to a nickname and piCommitCount parameter must be specified with a valid number (AUTOMATIC is not considered a valid option).

piXmlParse
Input. Type of parsing that should occur for XML documents. Valid values found in the db2ApiDf header file in the include directory, are:

DB2DMU_XMLPARSE_PRESERVE_WS
Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS
Whitespace should be stripped.

piXmlValidate
Input. Pointer to the db2DMUXmlValidate structure. Indicates that XML schema validation should occur for XML documents.

db2ImportOut data structure parameters

oRowsRead
Output. Number of records read from the file during import.

oRowsSkipped
Output. Number of records skipped before inserting or updating begins.

oRowsInserted
Output. Number of rows inserted into the target table.

oRowsUpdated
Output. Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).

oRowsRejected
Output. Number of records that could not be imported.

oRowsCommitted
Output. Number of records imported successfully and committed to the database.

db2DMUXmlMapSchema data structure parameters

iMapFromSchema
Input. The SQL identifier of the XML schema to map from.

iMapToSchema
Input. The SQL identifier of the XML schema to map to.

db2DMUXmlValidateXds data structure parameters

piDefaultSchema
Input. The SQL identifier of the XML schema that should be used for validation when an XDS does not contain an SCH attribute.

iNumIgnoreSchemas
Input. The number of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

piIgnoreSchemas
Input. The list of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

iNumMapSchemas
Input. The number of XML schemas that will be mapped during XML schema validation. The first schema in the schema map pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

piMapSchemas
Input. The list of XML schema pairs, where each pair represents a mapping of one schema to a different one. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

db2DMUXmlValidateSchema data structure parameters

piSchema
Input. The SQL identifier of the XML schema to use.

db2DMUXmlValidate data structure parameters

iUsing
Input. A specification of what to use to perform XML schema validation. Valid values found in the db2ApiDf header file in the include directory, are:

- **DB2DMU_XMLVAL_XDS**
  Validation should occur according to the XDS. This corresponds to the CLP ″XMLVALIDATE USING XDS″ clause.
- **DB2DMU_XMLVAL_SCHEMA**
  Validation should occur according to a specified schema. This corresponds to the CLP ″XMLVALIDATE USING SCHEMA″ clause.
- **DB2DMU_XMLVAL_SCHEMALOC_HINTS**
  Validation should occur according to schemaLocation hints found within the XML document. This corresponds to the ″XMLVALIDATE USING SCHEMALOCATION HINTS″ clause.

piXdsArgs
Input. Pointer to a db2DMUXmlValidateXds structure, representing arguments that correspond to the CLP ″XMLVALIDATE USING XDS″ clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_XDS.

piSchemaArgs
Input. Pointer to a db2DMUXmlValidateSchema structure, representing arguments that correspond to the CLP ″XMLVALIDATE USING SCHEMA″ clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_SCHEMA.

db2gImportStruct data structure specific parameters

iDataFileNameLen
Input. Specifies the length in bytes of piDataFileName parameter.

iFileTypeLen
Input. Specifies the length in bytes of piFileType parameter.

iMsgFileNameLen
Input. Specifies the length in bytes of piMsgFileName parameter.

Usage notes

Before starting an import operation, you must complete all table operations and release all locks in one of two ways:

- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

The import utility adds rows to the target table using the SQL INSERT statement.

The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.
- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a REPLACE or a REPLACE_CREATE operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the REPLACE or the REPLACE_CREATE option to rerun the whole import operation, or use INSERT with the iRestartcount parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the INSERT or the INSERT_UPDATE option. They are, however, performed if the *piCommitcount parameter is not zero. A full log results in a ROLLBACK.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

One cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:

1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions are preserved if the data was previously exported using SELECT *.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60 KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

Non-default values for piDataDescriptor, or specifying an explicit list of table columns in piLongActionString, makes importing to a remote database slower.

The database table or hierarchy must exist before data in the ASC, DEL, or WSF file formats can be imported; however, if the table does not already exist, IMPORT CREATE or IMPORT REPLACE_CREATE creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT CREATE can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the FORCEIN option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the FORCEIN option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the FORCEIN option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 for AIX clients.

For table objects on an 8KB page that are close to the limit of 1012 columns, import of PC/IXF data files may cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of DEL or ASC files.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (INSERT option) is supported. The restartcnt parameter, but not the commitcnt parameter, is also supported.

When using the CREATE option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than CREATE with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file. The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

On the Windows operating system:

- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Federated considerations

When using the db2Import API and the INSERT, UPDATE, or INSERT_UPDATE parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you wish to use when doing an import operation already exists.

Import sessions - CLP examples

Example 1
The following example shows how to import information frommyfile.ixf to the STAFF table:

```
   db2 import from myfile.ixf of ixf messages msg.txt insert into staff

SQL3150N  The H record in the PC/IXF file has product "DB2    01.00", date
"19970220", and time "140848".

SQL3153N  The T record in the PC/IXF file has name "myfile",
qualifier "        ", and source "            ".

SQL3109N  The utility is beginning to load data from file "myfile".

SQL3110N  The utility has completed processing.  "58" rows were read from the
input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "58".

SQL3222W  ...COMMIT of any database changes was successful.
```

```
SQL3149N  "58" rows were processed from the input file.  "58" rows were
successfully inserted into the table.  "0" rows were rejected.
```

Example 2
The following example shows how to import into a table that has identity columns:

TABLE1 has 4 columns:

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.

```
db2 import from datafile1.del of del replace into table1
```

To import DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:

```
db2 import from datafile1.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
db2 import from datafile1.del of del modified by identityignore
   replace into table1
```

To import DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

```
db2 import from datafile2.del of del replace into table1 (c1, c3, c4)
db2 import from datafile2.del of del modified by identitymissing
   replace into table1
```

If DATAFILE1 is imported into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be inserted, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

Example 3
The following example shows how to import into a table that has null indicators:

TABLE1 has 5 columns:

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:

- ELE1 positions 01 to 20
- ELE2 positions 21 to 22
- ELE5 positions 23 to 23
- ELE3 positions 24 to 27
- ELE4 positions 28 to 31
- ELE6 positions 32 to 32
- ELE6 positions 33 to 40

Data Records:

```
1...5....10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    wxyzN
Test data 4,5 and 6 WWN6789    Y
```

The following command imports records from ASCFILE1 into TABLE1:

```
db2 import from ascfile1 of asc
method L (1 20, 21 22, 24 27, 28 31)
null indicators (0, 0, 23, 32)
insert into table1 (col1, col5, col2, col3)
```

Note:

1. Since COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
2. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(.....)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
3. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
4. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Chapter 4. Load utility

Load overview

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data. The utility can handle most data types, including XML, large objects (LOBs), and user-defined types (UDTs). The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs. The load utility does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes).

The load process consists of four distinct phases (see Figure 3):

1. **Load**
   During the load phase, data is loaded into the table, and index keys and table statistics are collected, if necessary. *Save points*, or points of consistency, are established at intervals specified through the **SAVECOUNT** parameter in the LOAD command. Messages are generated, indicating how many input rows were successfully loaded at the time of the save point.
2. **Build**
   During the build phase, indexes are produced based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (if the STATISTICS USE PROFILE option was specified, and profile indicates collecting index statistics). The statistics are similar to those collected through the RUNSTATS command.
3. **Delete**
   During the delete phase, the rows that caused a unique or primary key violation are removed from the table. These deleted rows are stored in the load exception table, if one was specified.
4. **Index copy**
   During the index copy phase, the index data is copied from a system temporary table space to the original table space. This will only occur if a system temporary table space was specified for index creation during a load operation with the READ ACCESS option specified.

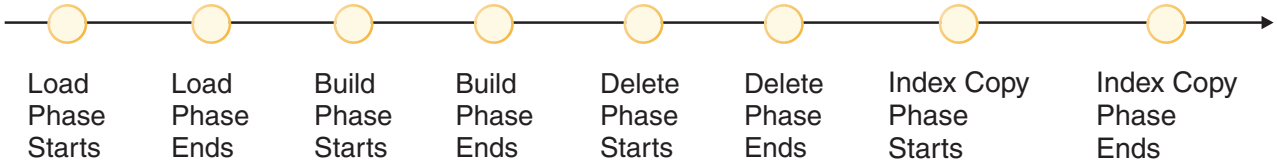


Figure 3. The Four Phases of the Load Process: Load, Build, Delete, and Index Copy

Note: After you invoke the load utility, you can use the LIST UTILITIES command to monitor the progress of the load operation.

The following information is required when loading data:

- The path and the name of the input file, named pipe, or device.
- The name or alias of the target table.
- The format of the input source. This format can be DEL, ASC, PC/IXF, or CURSOR.

© Copyright IBM Corp. 1993, 2008

- Whether the input data is to be appended to the table, or is to replace the existing data in the table.
- A message file name, if the utility is invoked through the application programming interface (API), db2Load.

Load modes

- **INSERT**
  In this mode, load appends input data to the table without making any changes to the existing data.
- **REPLACE**
  In this mode, load deletes existing data from the table and populates it with the input data.
- **RESTART**
  In this mode, an interrupted load is resumed. In most cases, the load is resumed from the phase it failed in. If that phase was the load phase, the load is resumed from the last successful consistency point.
- **TERMINATE**
  In this mode, a failed load operation is rolled back.

The options you can specify include:

- That the data to be loaded resides on the client, if the load utility is invoked from a remotely connected client. Note that XML and LOB data are always read from the server, even you specify the CLIENT option.
- The method to use for loading the data: column location, column name, or relative column position.
- How often the utility is to establish consistency points.
- The names of the table columns into which the data is to be inserted.
- Whether or not preexisting data in the table can be queried while the load operation is in progress.
- Whether the load operation should wait for other utilities or applications to finish using the table or force the other applications off before proceeding.
- An alternate system temporary table space in which to build the index.
- The paths and the names of the input files in which LOBs are stored.

  Note: The load utility does not honor the COMPACT lob option.
- A message file name. During load operations, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter.

  Note:

  1. You can only view the contents of a message file after the operation is finished. If you wish to view load messages while a load operation is running, you can use the LOAD QUERY command.
  2. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility.
- Whether column values being loaded have implied decimal points.
- Whether the utility should modify the amount of free space available after a table is loaded.
- Whether statistics are to be gathered during the load process. This option is only supported if the load operation is running in REPLACE mode. Statistics are

collected according to the profile defined for the table. The profile must be created by the RUNSTATS command before the LOAD command is executed. If the profile does not exist and the load operation is instructed to collect statistics according to the profile, the load will fail, and an error message will be returned.

If data is appended to a table, statistics are not collected. To collect current statistics on an appended table, invoke the RUNSTATS utility following completion of the load process. If gathering statistics on a table with a unique index, and duplicate keys are deleted during the delete phase, statistics are not updated to account for the deleted records. If you expect to have a significant number of duplicate records, do not collect statistics during the load operation. Instead, invoke the RUNSTATS utility following completion of the load process.

- Whether to keep a copy of the changes made. This is done to enable rollforward recovery of the database. This option is not supported if rollforward recovery is disabled for the database; that is, if the database configuration parameters *logarchmeth1* and *logarchmeth2* are set to OFF. If no copy is made, and rollforward recovery is enabled, the table space is left in Backup Pending state at the completion of the load operation.

  Logging is required for fully recoverable databases. The load utility almost completely eliminates the logging associated with the loading of data. In place of logging, you have the option of making a copy of the loaded portion of the table. If you have a database environment that allows for database recovery following a failure, you can do one of the following:

  – Explicitly request that a copy of the loaded portion of the table be made.
  – Take a backup of the table spaces in which the table resides immediately after the completion of the load operation.

  If the database configuration parameter *logindexbuild* is set, and if the load operation is invoked with the COPY YES recoverability option and the INCREMENTAL indexing option, the load logs all index modifications. The benefit of using these options is that when you roll forward through the log records for this load, you also recover the indexes (whereas normally the indexes are not recovered unless the load uses the REBUILD indexing mode).

  If you are loading a table that already contains data, and the database is non-recoverable, ensure that you have a backed-up copy of the database, or the table spaces for the table being loaded, before invoking the load utility, so that you can recover from errors.

  If you want to perform a sequence of multiple load operations on a recoverable database, the sequence of operations will be faster if you specify that each load operation is non-recoverable, and take a backup at the end of the load sequence, than if you invoke each of the load operations with the COPY YES option. You can use the NONRECOVERABLE option to specify that a load transaction is to be marked as non-recoverable, and that it will not be possible to recover it by a subsequent rollforward operation. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the rollforward operation is completed, such a table can only be dropped (see Figure 4 on page 128). With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation.

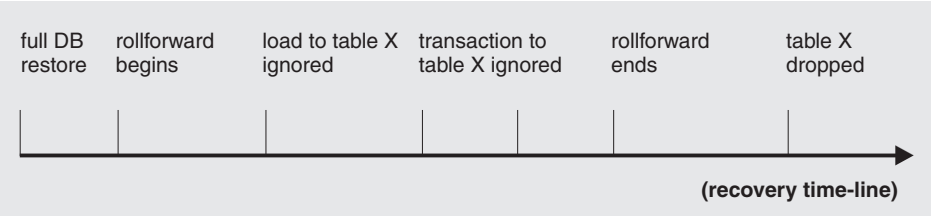


Figure 4. Non-recoverable Processing During a Roll Forward Operation

- The fully qualified path to be used when creating temporary files during a load operation. The name is specified by the TEMPFILES PATH parameter of the LOAD command. The default value is the database path. The path resides on the server machine, and is accessed by the DB2 instance exclusively. Therefore, any path name qualification given to this parameter must reflect the directory structure of the server, not the client, and the DB2 instance owner must have read and write permission on the path.

Privileges and authorities required to use load

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

To load data into a table, you must have one of the following:

- SYSADM authority
- DBADM authority
- LOAD authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
  - SELECT privilege on SYSCAT.TABLES is required in some cases where LOAD queries the catalog tables.

Since all load processes (and all DB2 server processes, in general), are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.

On Windows, and Windows.NET operating systems where DB2 is running as a Windows service, if you are loading data from files that reside on a network drive, you must configure the DB2 service to run under a user account that has read access to these files.

Note:

- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table.
- To load data into a table that has protected rows, the session authorization ID must have been granted a security label for write access that is part of the security policy protecting the table.

LOAD authority

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can use the LOAD command to load data into a table.

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can LOAD RESTART or LOAD TERMINATE if the previous load operation is a load to insert data.

Users having LOAD authority at the database level, as well as the INSERT and DELETE privileges on a table, can use the LOAD REPLACE command.

If the previous load operation was a load replace, the DELETE privilege must also have been granted to that user before the user can LOAD RESTART or LOAD TERMINATE.

If the exception tables are used as part of a load operation, the user must have INSERT privilege on the exception tables.

The user with this authority can perform QUIESCE TABLESPACES FOR TABLE, RUNSTATS, and LIST TABLESPACES commands.

Loading data

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data.

Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be loaded. Since the utility will issue a COMMIT statement, you should complete all transactions and release all locks by issuing either a COMMIT or a ROLLBACK statement before invoking the load utility. Data is loaded in the sequence that appears in the input file, except when using multidimensional clustering (MDC) tables, partitioned tables, or the `anyorder` file type modifier. If a particular sequence is desired, sort the data before attempting a load operation. If clustering is required, the data should be sorted on the clustering index prior to loading. When loading data into multidimensional clustered tables (MDC), sorting is not required prior to the load operation, and data is clustered according to the MDC table definition. When loading data into partitioned tables, sorting is not required prior to the load operation, and data is partitioned according to the table definition.

These are some of the restrictions that apply to the load utility (i.e., this list is not exhaustive):

- Loading data into nicknames is not supported.
- Loading data into typed tables, or tables with structured type columns, is not supported.
- Loading data into declared temporary tables is not supported.
- XML data can only be read from the server side; if you want to have the XML files read from the client, use the import utility.
- You cannot create or drop tables in a table space that is in Backup Pending state.
- You cannot load data into a database accessed through DB2 Connect or a server level prior to DB2 Version 2. Options that are only available with the current cannot be used with a server from the previous release.
- If an error occurs during a LOAD REPLACE operation, the original data in the table is lost. Retain a copy of the input data to allow the load operation to be restarted.
- Triggers are not activated on newly loaded rows. Business rules associated with triggers are not enforced by the load utility.
- Loading encrypted data is not supported.

These are some of the restrictions that apply to the load utility when loading into a partitioned table (i.e., this list is not exhaustive):

- Consistency points are not supported when the number of partitioning agents is greater than one.
- Loading data into a subset of data partitions while keeping the remaining data partitions fully online is not supported.
- The exception table used by a load operation or a set integrity pending operation cannot be partitioned.
- A unique index cannot be rebuilt when the load utility is running in insert mode or restart mode, and the load target table has any detached dependents.

The load utility can be invoked through the command line processor (CLP), the Load wizard in the Control Center, or an application programming interface (API) db2Load.

Using the Load wizard

1. From the Control Center, expand the object tree until you find the Tables folder.
2. Click on the Tables folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane).
3. In the contents pane, right-click on the table you want, and select Load from the pop-up menu. The Load wizard opens.
4. Specify the required information on each page of the wizard to successfully load your data.

Detailed information about the Load wizard is provided through its online help facility.

Issuing a LOAD command by using the CLP

The following is an example of a LOAD command issued through the CLP:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   insert into userid.staff copy yes use tsm data buffer 4000
```

In this example:

- Any warning or error messages are placed in the `staff.msgs` file.

- A copy of the changes made is stored in Tivoli® Storage Manager (TSM).
- Four thousand pages of buffer space are to be used during the load operation.

The following is another example of a LOAD command issued through the CLP:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   tempfiles path /u/myuser replace into staff
```

In this example:

- The table data is being replaced.
- The TEMPFILES PATH parameter is used to specify `/u/myuser` as the server path into which temporary files will be written.

Note: These examples use relative path names for the load input file. Relative path names are only allowed on calls from a client on the same database partition as the database. The use of fully qualified path names is recommended.

After you invoke the load utility, you can use the LIST UTILITIES command to monitor the progress of the load operation. In the case of a load operation performed in either INSERT mode, REPLACE mode, or RESTART mode, detailed progress monitoring support is available. Issue the LIST UTILITIES command with the SHOW DETAILS option to view detailed information about the current load phase. Details are not available for a load operation performed in TERMINATE mode. The LIST UTILITIES command will simply show that a load terminate utility is currently running.

A load operation maintains unique constraints, range constraints for partitioned tables, generated columns, and LBAC security rules. For all other constraints, the table is placed in the Set Integrity Pending state at the beginning of a load operation. After the load operation is complete, the SET INTEGRITY statement must be used to take the table out of Set Integrity Pending state.

Loading XML data

The load utility can be used for the efficient movement of large volumes of XML data into tables.

When loading data into an XML table column, you can use the XML FROM option to specify the paths of the input XML data file or files. For example, to load data from an XML file "/home/user/xmlpath/xmlfile1.xml" you could use the following command:

```
LOAD FROM data1.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

The delimited ASCII input file "data1.del" contains an XML data specifier (XDS) that describes the location of the XML data to load. For example, the following XDS describes an XML document at offset 123 bytes in file "xmldata.ext" that is 456 bytes in length:

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' />
```

Validating inserted documents against schemas

The XMLVALIDATE option allows XML documents to be validated against XML schemas as they are loaded. In the following example, incoming XML documents are validated against the schema identified by the XDS in the delimited ASCII input file "data2.del":

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T2
```

In this case, the XDS contains an SCH attribute with the fully qualified SQL identifier of the XML schema to use for validation, "S1.SCHEMA_A":

```
<XDS FIL='xmldata.ext' OFF='123' LEN='456' SCH='S1.SCHEMA_A' />
```

Specifying parse options

You can use the XMLPARSE option to specify whether whitespace in the loaded XML documents is preserved or stripped. In the following example, all loaded XML documents are validated against the schema with SQL identifier "S2.SCHEMA_A" and these documents are parsed with whitespace preserved:

```
LOAD FROM data2.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING SCHEMA S2.SCHEMA_A INSERT INTO USER.T1
```

Load considerations for partitioned tables

All of the existing load features are supported when the target table is partitioned with the exception of the following general restrictions:

- Consistency points are not supported when the number of partitioning agents is greater than one.
- Loading data into a subset of data partitions while the remaining data partitions remain fully online is not supported.
- The exception table used by a load operation cannot be partitioned.
- A unique index cannot be rebuilt when the load utility is running in insert mode or restart mode, and the load target table has any detached dependents.
- Similar to loading MDC tables, exact ordering of input data records is not preserved when loading partitioned tables. Ordering is only maintained within the cell or data partition.
- Load operations utilizing multiple formatters on each database partition only preserve approximate ordering of input records. Running a single formatter on each database partition, groups the input records by cell or table partitioning key. To run a single formatter on each database partition, explicitly request CPU_PARALLELISM of 1.

General load behavior

The load utility inserts data records into the correct data partition. There is no requirement to use an external utility, such as a splitter, to partition the input data before loading.

The load utility does not access any detached or attached data partitions. Data is inserted into visible data partitions only. Visible data partitions are neither attached nor detached. In addition, a load replace operation does not truncate detached or attached data partitions. Since the load utility acquires locks on the catalog system tables, the load utility waits for any uncommitted ALTER TABLE transactions. Such transactions acquire an exclusive lock on the relevant rows in the catalog tables, and the exclusive lock must terminate before the load operation can proceed. This means that there can be no uncommitted ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION transactions while load operation is running. Any input source records destined for an attached or detached data partition are rejected, and can be retrieved from the exception table if one is specified. An informational message is written to the message file to indicate some of the target table data partitions were in an attached or detached state. Locks on the relevant catalog table rows corresponding to the target table prevent users from changing

the partitioning of the target table by issuing any ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION operations while the load utility is running.

Handling of invalid rows
When the load utility encounters a record that does not belong to any of the visible data partitions the record is rejected and the load utility continues processing. The number of records rejected because of the range constraint violation is not explicitly displayed, but is included in the overall number of rejected records. Rejecting a record because of the range violation does not increase the number of row warnings. A single message (SQL0327N) is written to the load utility message file indicating that range violations are found, but no per-record messages are logged. In addition to all columns of the target table, the exception table includes columns describing the type of violation that had occurred for a particular row. Rows containing invalid data, including data that cannot be partitioned, are written to the dump file.

Because exception table inserts are expensive, you can control which constraint violations are inserted into the exception table. For instance, the default behavior of the load utility is to insert rows that were rejected because of a range constraint or unique constraint violation, but were otherwise valid, into the exception table. You can turn off this behavior by specifying, respectively, NORANGEEXC or NOUNIQUEEXC with the FOR EXCEPTION clause. If you specify that these constraint violations should not be inserted into the exception table, or you do not specify an exception table, information about rows violating the range constraint or unique constraint is lost.

History file
If the target table is partitioned, the corresponding history file entry does not include a list of the table spaces spanned by the target table. A different operation granularity identifier ('R' instead of 'T') indicates that a load operation ran against a partitioned table.

Terminating a load operation
Terminating a load replace completely truncates all visible data partitions, terminating a load insert truncates all visible data partitions to their lengths before the load. Indexes are invalidated during a termination of an online load operation that failed in the load copy phase. Indexes are also invalidated when terminating an offline load operation that touched the index (It is invalidated because the indexing mode is rebuild, or a key was inserted during incremental maintenance leaving the index in an inconsistent state). Loading data into multiple targets does not have any effect on load recovery operations except for the inability to restart the load operation from a consistency point taken during the load phase In this case, the SAVECOUNT load option is ignored if the target table is partitioned. This behavior is consistent with loading data into a MDC target table.

Generated columns
If a generated column is in any of the partitioning, dimension, or distribution keys, the `generatedoverride` file type modifier is ignored and the load utility generates values as if the `generatedignore` file type modifier is specified. Loading an incorrect generated column value in this case can place the record in the wrong physical location, such as the wrong data partition, MDC block or database partition. For example, once a record is on a wrong data partition, set integrity has to move it to a different physical location, which cannot be accomplished during online set integrity operations.

Data availability
The current online load algorithm extends to partitioned tables. An online load (ALLOW READ ACCESS) specified on the LOAD command allows concurrent readers to access the whole table, including both loading and non-loading data partitions.

Data partition states
After a successful load, visible data partitions might change to either or both Set Integrity Pending or Read Access Only table state, under certain conditions. Data partitions might be placed in these states if there are constraints on the table which the load operation cannot maintain. Such constraints might include check constraints and detached materialized query tables. A failed load operation leaves all visible data partitions in the Load Pending table state.

Error isolation
Error isolation at the data partition level is not supported. Isolating the errors means continuing a load on data partitions that did not run into an error and stopping on data partitions that did run into an error. Errors can be isolated between different database partitions, but the load utility cannot commit transactions on a subset of visible data partitions and roll back the remaining visible data partitions.

Other considerations

- Incremental indexing is not supported if any of the indexes are marked invalid. An index is considered invalid if it requires a rebuild or if detached dependents require validation with the SET INTEGRITY statement.
- Loading into tables partitioned using any combination of partitioned by range, distributed by hash, or organized by dimension algorithms is also supported.
- For log records which include the list of object and table space IDs affected by the load, the size of these log records (LOAD START and COMMIT (PENDING LIST)) could grow considerably and hence reduce the amount of active log space available to other applications.
- When a table is both partitioned and distributed, a partitioned database load might not affect all database partitions. Only the objects on the output database partitions are changed.
- During a load operation, memory consumption for partitioned tables increases with the number of tables. Note, that the total increase is not linear as only a small percentage of the overall memory requirement is proportional to the number of data partitions.

LBAC-protected data load considerations

For a successful load operation into a table with protected rows, you must have LBAC (label-based access control) credentials. You must also provide a valid security label, or a security label that can be converted to a valid label, for the security policy currently associated with the target table.

If you do not have valid LBAC credentials, the load fails and an error (SQLSTATE 42512) is returned. In cases where the input data does not contain a security label or that security label is not in its internal binary format, you can use several file type modifiers to allow your load to proceed.

When you load data into a table with protected rows, the target table has one column with a data type of DB2SECURITYLABEL. If the input row of data does not contain a value for that column, that row is rejected unless the `usedefaults` file

type modifier is specified in the load command, in which case the security label you hold for write access from the security policy protecting the table is used. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

When you load data into a table that has protected rows and the input data does include a value for the column with a data type of DB2SECURITYLABEL, the same rules are followed as when you insert data into that table. If the security label protecting the row being loaded (the one in that row of the data file) is one that you are able to write to, then that security label is used to protect the row. (In other words, it is written to the column that has a data type of DB2SECURITYLABEL.) If you are not able to write to a row protected by that security label, what happens depends on how the security policy protecting the source table was created:

- If the CREATE SECURITY POLICY statement that created the policy included the option RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL, the row is rejected.
- If the CREATE SECURITY POLICY statement did not include the option or if it instead included the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option, the security label in the data file for that row is ignored and the security label you hold for write access is used to protect that row. No error or warning is issued in this case. If you do not hold a security label for write access, the row is rejected and processing continues on to the next row.

Delimiter considerations
When loading data into a column with a data type of DB2SECURITYLABEL, the value in the data file is assumed by default to be the actual bytes that make up the internal representation of that security label. However, some raw data might contain newline characters which could be misinterpreted by the LOAD command as delimiting the row. If you have this problem, use the `delprioritychar` file type modifier to ensure that the character delimiter takes precedence over the row delimiter. When you use `delprioritychar`, any record or column delimiters that are contained within character delimiters are not recognized as being delimiters. Using the `delprioritychar` file type modifier is safe to do even if none of the values contain a newline character, but it does slow the load down slightly.

If the data being loaded is in ASC format, you might have to take an extra step in order to prevent any trailing white space from being included in the loaded security labels and security label names. ASCII format uses column positions as delimiters, so this might occur when loading into variable-length fields. Use the `striptblanks` file type modifier to truncate any trailing blank spaces.

Nonstandard security label values
You can also load data files in which the values for the security labels are strings containing the values of the components in the security label, for example, *S:(ALPHA,BETA)*. To do so you must use the file type modifier `seclabelchar`. When you use `seclabelchar`, a value for a column with a data type of DB2SECURITYLABEL is assumed to be a string constant containing the security label in the string format for security labels. If a string is not in the proper format, the row is not inserted and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table, the row is not inserted and a warning (SQLSTATE 01H53) is returned.

You can also load a data file in which the values of the security label column are security label names. To load this sort of file you must use the file type modifier `seclabelname`. When you use `seclabelname`, all values for columns with a data type of DB2SECURITYLABEL are assumed to be string constants containing the names of existing security labels. If no security label exists with the indicated name for the security policy protecting the table, the row is not loaded and a warning (SQLSTATE 01H53) is returned.

Rejected rows
Rows that are rejected during the load are sent to either a dumpfile or an exception table (if they are specified in the LOAD command), depending on the reason why the rows were rejected. Rows that are rejected due to parsing errors are sent to the dumpfile. Rows that violate security policies are sent to the exception table.

Examples

For all examples, the input data file myfile.del is in DEL format. All are loading data into a table named REPS, which was created with this statement:

```
create table reps (row_label db2securitylabel,
id integer,
name char(30))
security policy data_access_policy
```

For this example, the input file is assumed to contain security labels in the default format:

```
db2 load from myfile.del of del modified by delprioritychar insert into reps
```

For this example, the input file is assumed to contain security labels in the security label string format:

```
db2 load from myfile.del of del modified by seclabelchar insert into reps
```

For this example, the input file is assumed to contain security labels names for the security label column:

```
db2 load from myfile.del of del modified by seclabelname insert into reps
```

Identity column load considerations

The load utility can be used to load data into a table containing an identity column whether or not the input data has identity column values.

If no identity-related file type modifiers are used, the utility works according to the following rules:

- If the identity column is GENERATED ALWAYS, an identity value is generated for a table row whenever the corresponding row in the input file is missing a value for the identity column, or a NULL value is explicitly given. If a non-NULL value is specified for the identity column, the row is rejected (SQL3550W).
- If the identity column is GENERATED BY DEFAULT, the load utility makes use of user-supplied values, if they are provided; if the data is missing or explicitly NULL, a value is generated.

The load utility does not perform any extra validation of user-supplied identity values beyond what is normally done for values of the identity column's data type (that is, SMALLINT, INT, BIGINT, or DECIMAL). Duplicate values are not reported.

In most cases the load utility cannot guarantee that identity column values are assigned to rows in the same order that these rows appear in the data file. Because the assignment of identity column values is managed in parallel by the load utility, those values are assigned in arbitrary order. The exceptions to this are as follows:

- In single-partition databases, rows are not processed in parallel when CPU_PARALLELISM is set to 1. In this case, identity column values are implicitly assigned in the same order that rows appear in the data file parameter.
- In multi-partition databases, identity column values are assigned in the same order that the rows appear in the data file if the identity column is in the distribution key and if there is a single partitioning agent (that is, if you do not specify multiple partitioning agents or the `anyorder` file type modifier).

When loading a table in a partitioned database where the table has an identity column in the partitioning key and the `identityoverride` modifier is not specified, the SAVECOUNT option cannot be specified. When there is an identity column in the partitioning key and identity values are being generated, restarting a load from the load phase on at least one database partition requires restarting the whole load from the beginning of the load phase, which means that there can't be any consistency points.

Note: A load RESTART operation is not permitted if all of the following criteria are met:

- The table being loaded is in a partitioned database environment, and it contains at least one identity column that is either in the distribution key or is referenced by a generated column that is part of the distribution key.
- The `identityoverride` modifier is not specified.
- The previous load operation that failed included loading database partitions that failed after the load phase.

A load TERMINATE or REPLACE operation should be issued instead.

There are three mutually exclusive ways you can simplify the loading of data into tables that contain an identity column: the `identitymissing`, the `identityignore`, and the `identityoverride` file type modifiers.

Loading data without identity columns

The `identitymissing` modifier makes loading a table with an identity column more convenient if the input data file does not contain any values (not even NULLS) for the identity column. For example, consider a table defined with the following SQL statement:

```
create table table1 (c1 varchar(30),
                     c2 int generated by default as identity,
                     c3 decimal(7,2),
                     c4 char(1))
```

If you want to load TABLE1 with data from a file (`load.del`) that has been exported from a table that does not have an identity column, see the following example:

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

One way to load this file would be to explicitly list the columns to be loaded through the LOAD command as follows:

```
db2 load from load.del of del replace into table1 (c1, c3, c4)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of loading the file is to use the `identitymissing` file type modifier as follows:

```
db2 load from load.del of del modified by identitymissing
   replace into table1
```

This command would result in the three columns in the data file being loaded into c1, c3, and c4 of TABLE1. A value will be generated for each row in c2.

Loading data with identity columns
The `identityignore` modifier indicates to the load utility that even though the input data file contains data for the identity column, the data should be ignored, and an identity value should be generated for each row. For example, a user might want to load TABLE1, as defined above, from a data file (`load.del`) containing the following data:

```
Robert, 1, 45.2, J
Mike, 2, 76.9, K
Leo, 3, 23.4, I
```

If the user-supplied values of 1, 2, and 3 are not used for the identity column, you can issue the following LOAD command:

```
db2 load from load.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The `identityignore` modifier simplifies the syntax as follows:

```
db2 load from load.del of del modified by identityignore
   replace into table1
```

Loading data with user-supplied values
The `identityoverride` modifier is used for loading user-supplied values into a table with a GENERATED ALWAYS identity column. This can be quite useful when migrating data from another database system, and the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data (or NULL data) for the identity column are rejected (SQL3116W). You should also note that when using this modifier, it is possible to violate the uniqueness property of GENERATED ALWAYS columns.In this situation, perform a load TERMINATE operation, followed by a subsequent load INSERT or REPLACE operation.

Generated column load considerations

You can load data into a table containing (nonidentity) generated columns whether or not the input data has generated column values. The load utility generates the column values.

If no generated column-related file type modifiers are used, the load utility works according to the following rules:

- Values are created for generated columns when the corresponding row of the data file is missing a value for the column or a NULL value is supplied. If a non-NULL value is supplied for a generated column, the row is rejected (SQL3550W).

- If a NULL value is created for a generated column that is not nullable, the entire row of data is rejected (SQL0407N). This could occur if, for example, a non-nullable generated column is defined as the sum of two table columns that include NULL values in the data file.

There are three mutually exclusive ways you can simplify the loading of data into tables that contain a generated column: the `generatedmissing`, the `generatedignore`, and the `generatedoverride` file type modifiers.

Loading data without generated columns

The `generatedmissing` modifier makes loading a table with generated columns more convenient if the input data file does not contain any values (not even NULLS) for all generated columns present in the table. For example, consider a table defined with the following SQL statement:

```
CREATE TABLE table1 (c1 INT,
                     c2 INT,
                     g1 INT GENERATED ALWAYS AS (c1 + c2),
                     g2 INT GENERATED ALWAYS AS (2 * c1),
                     c3 CHAR(1))
```

If you want to load TABLE1 with data from a file (`load.del`) that has been exported from a table that does not have any generated columns, see the following example:

```
1, 5, J
2, 6, K
3, 7, I
```

One way to load this file would be to explicitly list the columns to be loaded through the LOAD command as follows:

```
DB2 LOAD FROM load.del of del REPLACE INTO table1 (c1, c2, c3)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of loading the file is to use the `generatedmissing` file type modifier as follows:

```
DB2 LOAD FROM load.del of del MODIFIED BY generatedmissing
   REPLACE INTO table1
```

This command will result in the three columns of data file being loaded into c1, c2, and c3 of TABLE1. Due to the `generatedmissing` modifier, values for columns g1 and g2 of TABLE1 will be generated automatically and will not map to any of the data file columns.

Loading data with generated columns

The `generatedignore` modifier indicates to the load utility that even though the input data file contains data for all generated columns present in the target table, the data should be ignored, and the computed values should be loaded into each generated column. For example, if you want to load TABLE1, as defined above, from a data file (`load.del`) containing the following data:

```
1, 5, 10, 15, J
2, 6, 11, 16, K
3, 7, 12, 17, I
```

The user-supplied, non-NULL values of 10, 11, and 12 (for g1), and 15, 16, and 17 (for g2) result in the row being rejected (SQL3550W) if no generated-column related file type modifiers are used. To avoid this, the user could issue the following LOAD command:

```
DB2 LOAD FROM load.del of del method P(1, 2, 5)
   REPLACE INTO table1 (c1, c2, c3)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The generatedignore modifier simplifies the syntax as follows:

```
DB2 LOAD FROM load.del of del MODIFIED BY generatedignore
   REPLACE INTO table1
```

This command will result in the columns of data file being loaded into c1 (with the data 1, 2, 3), c2 (with the data 5,6,7), and c3 (with the data J, K, I) of TABLE1. Due to the generatedignore modifier, values for columns g1 and g2 of TABLE1 will be generated automatically and the data file columns (10, 11, 12 and 15, 16, 17) will be ignored.

Loading data with user-supplied values

The generatedoverride modifier is used for loading user-supplied values into a table with generated columns. This can be useful when migrating data from another database system, or when loading a table from data that was recovered using the RECOVER DROPPED TABLE option of the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data (or NULL data) for non-nullable generated columns are rejected (SQL3116W).

When this modifier is used, the table is placed in the Set Integrity Pending state after the load operation. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command:

```
SET INTEGRITY FOR table-name GENERATED COLUMN IMMEDIATE
  UNCHECKED
```

To take the table out of the Set Integrity Pending state and force verification of the user-supplied values, issue the following command:

```
SET INTEGRITY FOR table-name IMMEDIATE CHECKED
```

If a generated column is in any of the partitioning, dimension, or distribution keys, the generatedoverride modifier is ignored and the load utility generates values as if the generatedignore modifier is specified. This is done to avoid a scenario where a user-supplied generated column value conflicts with its generated column definition, which would place the resulting record in the wrong physical location, such as the wrong data partition, MDC block, or database partition.

Note: There is one case where load does NOT support generating column values: when one of the generated column expressions contains a user-defined function that is FENCED. If you attempt to load into such a table the load operation fails. However, you can provide your own values for these types of generated columns by using the generatedoverride file type modifier.

Considerations for using a Version 7 or earlier client with a Version 8 or later server

If you initiate a load operation between a Version 7 or earlier client and a Version 8 or later server, the load utility places tables with generated columns in the Set Integrity Pending state. If a table has been placed in Set Integrity Pending state because a Version 7 or earlier client was used to load data into a table with generated columns, issue the following statement to remove that state and force the generation of values:

```
SET INTEGRITY FOR table-name IMMEDIATE CHECKED FORCE GENERATED;
```

Moving data using the CURSOR file type

By specifying the CURSOR file type when using the LOAD command, you can load the results of an SQL query directly into a target table without creating an intermediate exported file.

Additionally, you can load data from another database by referencing a nickname within the SQL query, by using the DATABASE option within the DECLARE CURSOR statement, or by using the sqlu_remotefetch_entry media entry when using the API interface.

There are three approaches for moving data using the CURSOR file type. The first approach uses the Command Line Processor (CLP), the second the API, and the third uses the ADMIN_CMD procedure. The key differences between the CLP and the ADMIN_CMD procedure are outlined in the following table.

Table 26. . Differences between the CLP and ADMIN_CMD procedure.

Differences	**CLP**	**ADMIN_CMD_procedure**
Syntax	The query statement as well as the source database used by the cursor are defined outside of the LOAD command using a DECLARE CURSOR statement.	The query statement as well as the source database used by the cursor is defined within the LOAD command using the LOAD from (DATABASE database-alias query-statement)
User authorization for accessing a different database	If the data is in a different database than the one you currently connect to, the DATABASE keyword must be used in the DECLARE CURSOR statement. You can specify the user id and password in the same statement as well. If the user id and password are not specified in the DECLARE CURSOR statement, the user id and password explicitly specified for the source database connection are used to access the target database.	If the data is in a different database than the one you are currently connected to, the DATABASE keyword must be used in the LOAD command before the query statement. The user id and password explicitly specified for the source database connection are required to access the target database. You cannot specify a userid or password for the source database. Therefore, if no userid and password were specified when the connection to the target database was made, or the userid and password specified cannot be used to authenticate against the source database, the ADMIN_CMD procedure cannot be used to perform the load.

To execute a LOAD FROM CURSOR operation from the CLP, a cursor must first be declared against an SQL query. Once this is declared, you can issue the LOAD command using the declared cursor's name as the *cursorname* and CURSOR as the file type.

For example:

1. Suppose a source and target table both reside in the same database with the following definitions:

   Table ABC.TABLE1 has 3 columns:

   - ONE INT
   - TWO CHAR(10)
   - THREE DATE

   Table ABC.TABLE2 has 3 columns:

   - ONE VARCHAR
   - TWO INT
   - THREE DATE

   Executing the following CLP commands will load all the data from ABC.TABLE1 into ABC.TABLE2:

   ```
   DECLARE mycurs CURSOR FOR SELECT TWO, ONE, THREE FROM abc.table1
      LOAD FROM mycurs OF cursor INSERT INTO abc.table2
   ```

   Note: The above example shows how to load from an SQL query through the CLP. However, loading from an SQL query can also be accomplished through the db2Load API. Define the *piSourceList* of the sqlu_media_list structure to use the *sqlu_statement_entry* structure and SQLU_SQL_STMT media type and define the *piFileType* value as SQL_CURSOR.
2. Suppose the source and target tables reside in different databases with the following definitions:

Table ABC.TABLE1 in database 'dbsource' has 3 columns:

- ONE INT
- TWO CHAR(10)
- THREE DATE

Table ABC.TABLE2 in database 'dbtarget' has 3 columns:

- ONE VARCHAR
- TWO INT
- THREE DATE

Provided that you have enabled federation and cataloged the data source ('dsdbsource'), you can declare a nickname against the source database, then declare a cursor against this nickname, and invoke the LOAD command with the FROM CURSOR option, as demonstrated in the following example:

```
CREATE NICKNAME myschema1.table1 FOR dsdbsource.abc.table1
DECLARE mycurs CURSOR FOR SELECT TWO,ONE,THREE FROM myschema1.table1
LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

Or, you can use the DATABASE option of the DECLARE CURSOR statement, as demonstrated in the following example:

```
DECLARE mycurs CURSOR DATABASE dbsource USER dsciaraf USING mypasswd
FOR SELECT TWO,ONE,THREE FROM abc.table1
LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

Using the DATABASE option of the DECLARE CURSOR statement (also known as the remotefetch media type when using the Load API) has some benefits over the nickname approach:

Performance

Fetching of data using the remotefetch media type is tightly integrated within a load operation. There are fewer layers of transition to fetch a record compared to the nickname approach. Additionally, when source and target tables are distributed identically in a multi-partition database, the load utility can parallelize the fetching of data, which can further improve performance.

Ease of use

There is no need to enable federation, define a remote datasource, or declare a nickname. Specifying the DATABASE option (and the USER and USING options if necessary) is all that is required.

While this method can be used with cataloged databases, the use of nicknames provides a robust facility for fetching from various data sources which cannot simply be cataloged.

To support this remotefetch functionality, the load utility makes use of infrastructure which supports the SOURCEUSEREXIT facility. The load utility spawns a process which executes as an application to manage the connection to the source database and perform the fetch. This application is associated with its own transaction and is not associated with the transaction under which the load utility is running.

Note:

1. The previous example shows how to load from an SQL query against a cataloged database through the CLP using the DATABASE option of the DECLARE CURSOR statement. However, loading from an SQL query against a cataloged database can also be done through the db2Load API, by defining the *piSourceList* and *piFileTypevalues* of the *db2LoadStruct* structure to use the sqlu_remotefetch_entry media entry and SQLU_REMOTEFETCH media type respectively.
2. As demonstrated in the previous example, the source column types of the SQL query do not need to be identical to their target column types, although they do have to be compatible.

Restrictions

When loading from a cursor defined using the DATABASE option (or equivalently when using the sqlu_remotefetch_entry media entry with the db2Load API), the following restrictions apply:

1. The SOURCEUSEREXIT option cannot be specified concurrently.
2. The METHOD N option is not supported.
3. The `usedefaults` file type modifier is not supported.

Propagating dependent immediate staging tables

If the table being loaded is an underlying table of a staging table with the immediate propagate attribute, and if the load operation is done in insert mode, the subsequent propagation into the dependent immediate staging tables is incremental.

During incremental propagation, the rows corresponding to the appended rows in the underlying tables are appended into the staging tables. Incremental propagation is faster in the case of large underlying tables with small amounts of appended data. Performance is also improved if the staging table is used to refresh

its dependent deferred materialized query table. There are cases in which incremental propagation is not allowed, and the staging table is marked incomplete. That is, the staging byte of the CONST_CHECKED column has a value of F. In this state, the staging table can not be used to refresh its dependent deferred materialized query table, and a full refresh is required in the materialized query table maintenance process.

If a table is in incomplete state and the INCREMENTAL option has been specified, but incremental propagation of the table is not possible, an error is returned. If any of the following have taken place, the system turns off immediate data propagation and sets the table state to incomplete:

- A load replace operation has taken place on an underlying table of the staging table, or the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated after the last integrity check on the underlying table.
- The dependent materialized query table of the staging table, or the staging table has been loaded in REPLACE or INSERT mode.
- An underlying table has been taken out of Set Integrity Pending state before the staging table has been propagated by using the FULL ACCESS option during integrity checking.
- An underlying table of the staging table has been checked for integrity non-incrementally.
- The table space containing the staging table or its underlying table has been rolled forward to a point in time, and the staging table and its underlying table reside in different table spaces.

If the staging table has a `W` value in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and the NOT INCREMENTAL option is not specified, incremental propagation to the staging table takes place and the CONST_CHECKED column of SYSCAT.TABLES is marked as `U` to indicate that not all data has been verified by the system.

The following example illustrates a load insert operation into the underlying table UT1 of staging table G1 and its dependent deferred materialized query table AST1. In this scenario, both the integrity checking for UT1 and the refreshing of AST1 are processed incrementally:

```
LOAD FROM IMTFILE1.IXF of IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF of IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1,G1 IMMEDIATE CHECKED;

REFRESH TABLE AST1 INCREMENTAL;
```

Refreshing dependent immediate materialized query tables

If the underlying table of an immediate refresh materialized query table is loaded using the INSERT option, executing the SET INTEGRITY statement on the dependent materialized query tables defined with REFRESH IMMEDIATE results in an incremental refresh of the materialized query table.

During an incremental refresh, the rows corresponding to the appended rows in the underlying tables are updated and inserted into the materialized query tables. Incremental refresh is faster in the case of large underlying tables with small amounts of appended data. There are cases in which incremental refresh is not allowed, and full refresh (that is, recomputation of the materialized query table definition query) is used.

When the INCREMENTAL option is specified, but incremental processing of the materialized query table is not possible, an error is returned if:

- A load replace operation has taken place into an underlying table of the materialized query table or the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated since the last integrity check on the underlying table.
- The materialized query table has been loaded (in either REPLACE or INSERT mode).
- An underlying table has been taken out of Set Integrity Pending state before the materialized query table is refreshed by using the FULL ACCESS option during integrity checking.
- An underlying table of the materialized query table has been checked for integrity non-incrementally.
- The materialized query table was in Set Integrity Pending state before migration.
- The table space containing the materialized query table or its underlying table has been rolled forward to a point in time, and the materialized query table and its underlying table reside in different table spaces.

If the materialized query table has one or more `W` values in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and if the NOT INCREMENTAL option is not specified in the SET INTEGRITY statement, the table is incrementally refreshed and the CONST_CHECKED column of SYSCAT.TABLES is marked `U` to indicate that not all data has been verified by the system.

The following example illustrates a load insert operation into the underlying table UTI of the materialized query table AST1. UT1 is checked for data integrity and is placed in the no data movement mode. UT1 is put back into full access state once the incremental refresh of AST1 is complete. In this scenario, both the integrity checking for UT1 and the refreshing of AST1 are processed incrementally.

```
LOAD FROM IMTFILE1.IXF of IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF of IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1 IMMEDIATE CHECKED;
REFRESH TABLE AST1;
```

Multidimensional clustering considerations

The following restrictions apply when loading data into multidimensional clustering (MDC) tables:

- The SAVECOUNT option of the LOAD command is not supported.
- The `totalfreespace` file type modifier is not supported since these tables manage their own free space.
- The `anyorder` file type modifier is required for MDC tables. If a load is executed into an MDC table without the `anyorder` modifier, it will be explicitly enabled by the utility.

When using the LOAD command with an MDC table, violations of unique constraints are be handled as follows:

- If the table included a unique key prior to the load operation and duplicate records are loaded into the table, the original record remains and the new records are deleted during the delete phase.
- If the table did not include a unique key prior to the load operation and both a unique key and duplicate records are loaded into the table, only one of the records with the unique key is loaded and the others are deleted during the delete phase.

Note: There is no explicit technique for determining which record is loaded and which is deleted.

Performance Considerations

To improve the performance of the load utility when loading MDC tables, the *util_heap_sz* database configuration parameter value should be increased. The mdc-load algorithm performs significantly better when more memory is available to the utility. This reduces disk I/O during the clustering of data that is performed during the load phase. When the DATA BUFFER option of the LOAD command is specified, its value should also be increased. If the LOAD command is being used to load several MDC tables concurrently, *util_heap_sz* should be increased accordingly.

MDC load operations always have a build phase since all MDC tables have block indexes.

During the load phase, extra logging for the maintenance of the block map is performed. There are approximately two extra log records per extent allocated. To ensure good performance, the *logbufsz* database configuration parameter should be set to a value that takes this into account.

A system temporary table with an index is used to load data into MDC tables. The size of the table is proportional to the number of distinct cells loaded. The size of each row in the table is proportional to the size of the MDC dimension key. To minimize disk I/O caused by the manipulation of this table during a load operation, ensure that the buffer pool for the temporary table space is large enough.

Moving data using a customized application (user exit)

The load SOURCEUSEREXIT option provides a facility through which the load utility can execute a customized script or executable, referred to herein as a *user exit*.

The purpose of the user exit is to populate one or more named pipes with data that is simultaneously read from by the load utility. In a multi-partition database, multiple instances of the user exit can be invoked concurrently to achieve parallelism of the input data.

As Figure 5 on page 147 shows, the load utility creates a one or more named pipes and spawns a process to execute your customized executable. Your user exit feeds data into the named pipe(s) while the load utility simultaneously reads.

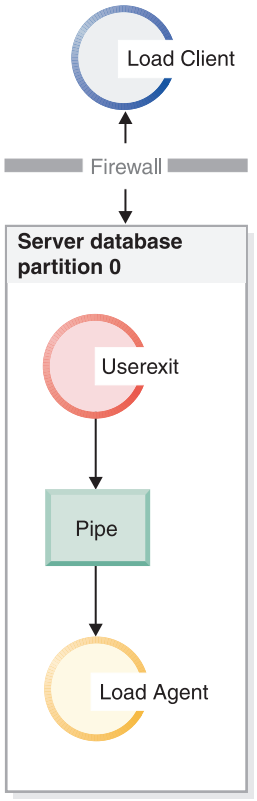


Figure 5. The load utility reads from the pipe and processes the incoming data.

The data fed into the pipe must reflect the load options specified, including the file type and any file type modifiers. The load utility does not directly read the data files specified. Instead, the data files specified are passed as arguments to your user exit when it is executed.

Invoking your user exit

The user exit must reside in the bin subdirectory of the DB2 installation directory (often known as sqllib). The load utility invokes the user exit executable with the following command line arguments:

```
<base pipename> <number of source media>
<source media 1> <source media 2> ... <user exit ID>
<number of user exits> <database partition number>
```

Where:

<base pipename >
Is the base name for named-pipes that the load utility creates and reads data from. The utility creates one pipe for every source file provided to the LOAD command, and each of these pipes is appended with `.xxx`, where `xxx` is the index of the source file provided. For example, if there are 2 source files provided to the LOAD command, and the <base pipename> argument passed to the user exit is `pipe123`, then the two named pipes that your user exit should feed with data are `pipe123.000` and `pipe123.001`. In a partitioned database environment, the load utility appends the database partition (DBPARTITION) number `.yyy` to the base pipe name, resulting in the pipe name `pipe123.xxx.yyy`..

<number of source media>
Is the number of media arguments which follow.

<source media 1> <source media 2> ...
Is the list of one or more source files specified in the LOAD command. Each source file is placed inside double quotation marks.

<user exit ID>
Is a special value useful when the PARALLELIZE option is enabled. This integer value (from 1 to N, where N is the total number of user exits being spawned) identifies a particular instance of a running user exit. When the PARALLELIZE option is not enabled, this value defaults to 1.

<number of user exits>
Is a special value useful when the PARALLELIZE option is enabled. This value represents the total number of concurrently running user exits. When the PARALLELIZE option is not enabled, this value defaults to 1.

<database partition number>
Is a special value useful when the PARALLELIZE option is enabled. This is the database partition (DBPARTITION) number on which the user exit is executing. When the PARALLELIZE option is not enabled, this value defaults to 0.

Additional options and features

The following section describes additional SOURCEUSEREXIT facility options:

REDIRECT
This option allows you to pass data into the STDIN handle or capture data from the STDOUT and STDERR handles of the user exit process.

INPUT FROM BUFFER <buffer>
Allows you to pass information directly into the STDIN input stream of your user exit. After spawning the process which executes the user exit, the load utility acquires the file-descriptor to the STDIN of this new process and passes in the buffer provided. The user exit reads from STDIN to acquire the information. The load utility simply sends the contents of <buffer> to the user exit using STDIN and does not interpret or modify its contents. For example, if your user exit is designed to read two values from STDIN, an eight-byte userid and an eight-byte password, your user exit executable written in C might contain the following lines:

```
rc = read (stdin, pUserID, 8);
rc = read (stdin, pPasswd, 8);
```

A user could pass this information using the INPUT FROM BUFFER option as shown in the following LOAD command:

```
LOAD FROM myfile1 OF DEL INSERT INTO table1
SOURCEUSEREXIT myuserexit1 REDIRECT INPUT FROM BUFFER myuseridmypasswd
```

Note: The load utility limits the size of <buffer> to the maximum size of a LOB value. However, from within the command line processor (CLP), the size of <buffer> is restricted to the maximum size of a CLP statement. From within CLP, it is also recommended that <buffer> contain only traditional ASCII characters. These issues can be avoided if the load utility is invoked using the db2Load API, or if the INPUT FROM FILE option is used instead.

INPUT FROM FILE <filename>
Allows you to pass the contents of a client side file directly into the STDIN input stream of your user exit. This option is almost identical to the INPUT FROM BUFFER option, however this option avoids the potential CLP limitation. The filename must be a fully qualified client side file and must not be larger than the maximum size of a LOB value.

OUTPUT TO FILE <filename>
Allows you to capture the STDOUT and STDERR streams from your user exit process into a server side file. After spawning the process which executes the user exit executable, the load utility redirects the STDOUT and STDERR handles from this new process into the filename specified. This option is useful for debugging and logging errors and activity within your user exit. The filename must be a fully qualified server side file. When the PARALLELIZE option is enabled, one file exists per user exit and each file appends a three-digit numeric identifier, such as *filename.000*.

PARALLELIZE
This option can increase the throughput of data coming into the load utility by invoking multiple user exit processes simultaneously. This option is only applicable to a multi-partition database. The number of user exit instances invoked is equal to the number of distribution agents if data is to be distributed across multiple database partitions during the load operation, otherwise it is equal to the number of loading agents.

The <user exit ID>, <number of user exits>, and <database partition number> arguments passed into each user exit reflect the unique identifier (1 to N), the total number of user exits (N), and the database partition (DBPARTITION) number on which the user exit instance is running, respectively. You should ensure that any data written to the named pipe by each user exit process is not duplicated by the other concurrent processes. While there are many ways your user exit application might accomplish this, these values could be helpful to ensure data is not duplicated. For example, if each record of data contains a unique integer column value, your user exit application could use the <user exit ID> and <number of user exits> values to ensure that each user exit instance returns a unique result set into its named pipe. Your user exit application might use the **MODULUS** property in the following way:

```
i = <user exit ID>
N = <number of user exits>

foreach record
{
   if ((unique-integer MOD N) == i)
   {
     write this record to my named-pipe
   }
}
```

The number of user exit processes spawned depends on the distribution mode specified for database partitioning:

1. As Figure 6 on page 150 shows, one user exit process is spawned for every pre-distribution agent when PARTITION_AND_LOAD (default) or PARTITION_ONLY without PARALLEL is specified. .

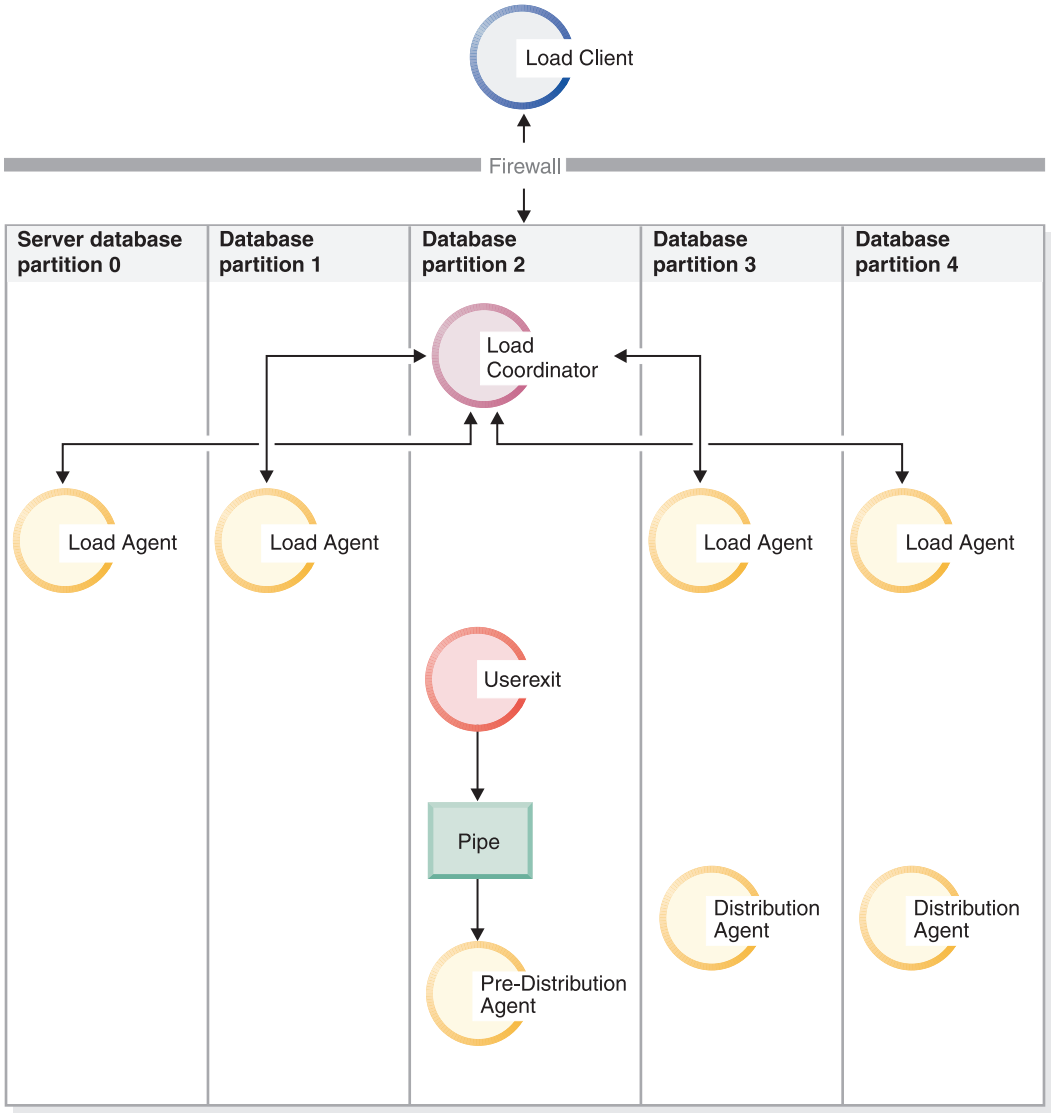


Figure 6. The various tasks performed when PARTITION_AND_LOAD (default) or PARTITION_ONLY without PARALLEL is specified.

2. As Figure 7 on page 151 shows, one user exit process is spawned for every distribution agent when PARTITION_AND_LOAD (default) or PARTITION_ONLY with PARALLEL is specified.

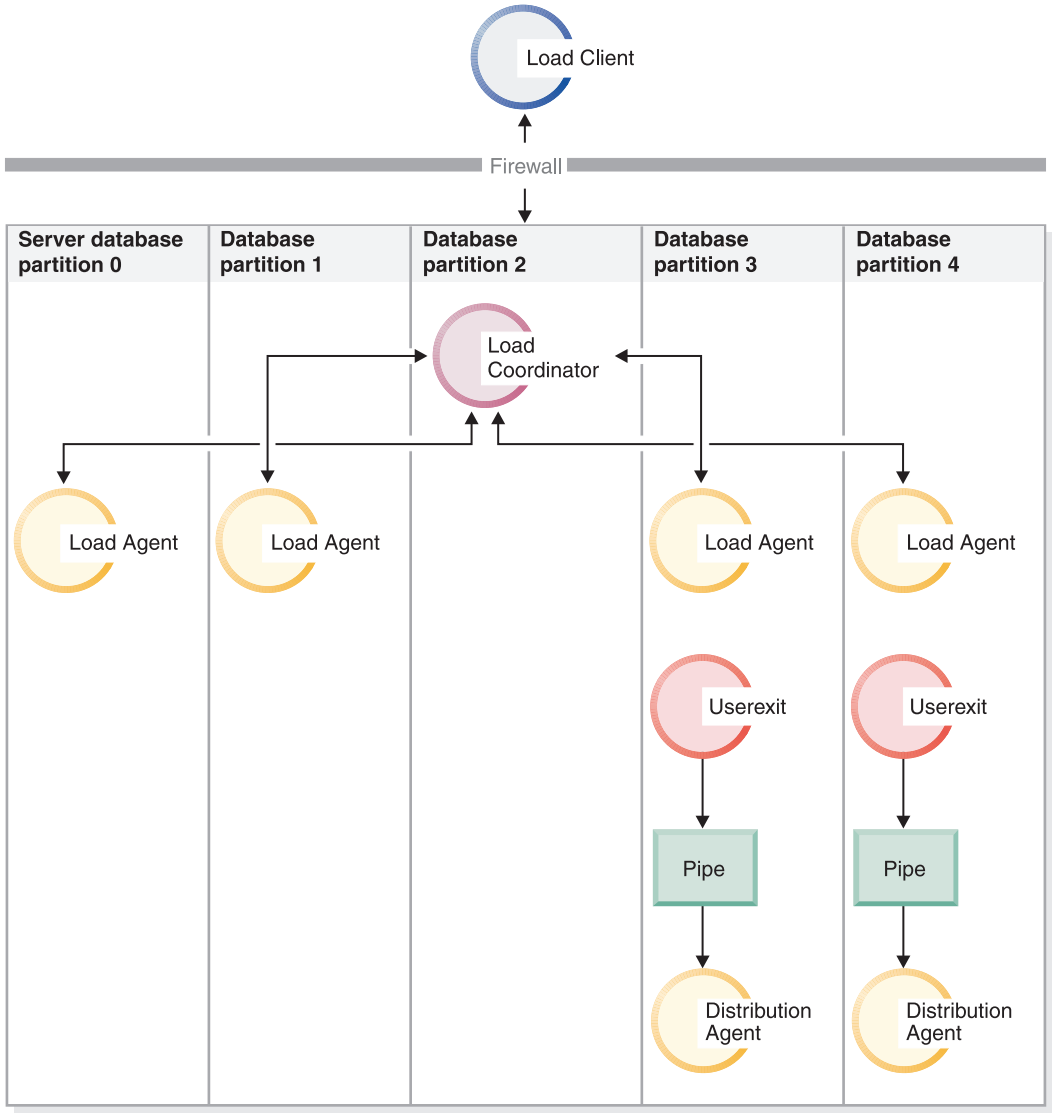


Figure 7. The various tasks performed when PARTITION_AND_LOAD (default) or PARTITION_ONLY with PARALLEL is specified.

3. As Figure 8 on page 152 shows, one user exit process is spawned for every load agent when LOAD_ONLY or LOAD_ONLY_VERIFY_PART is specified.

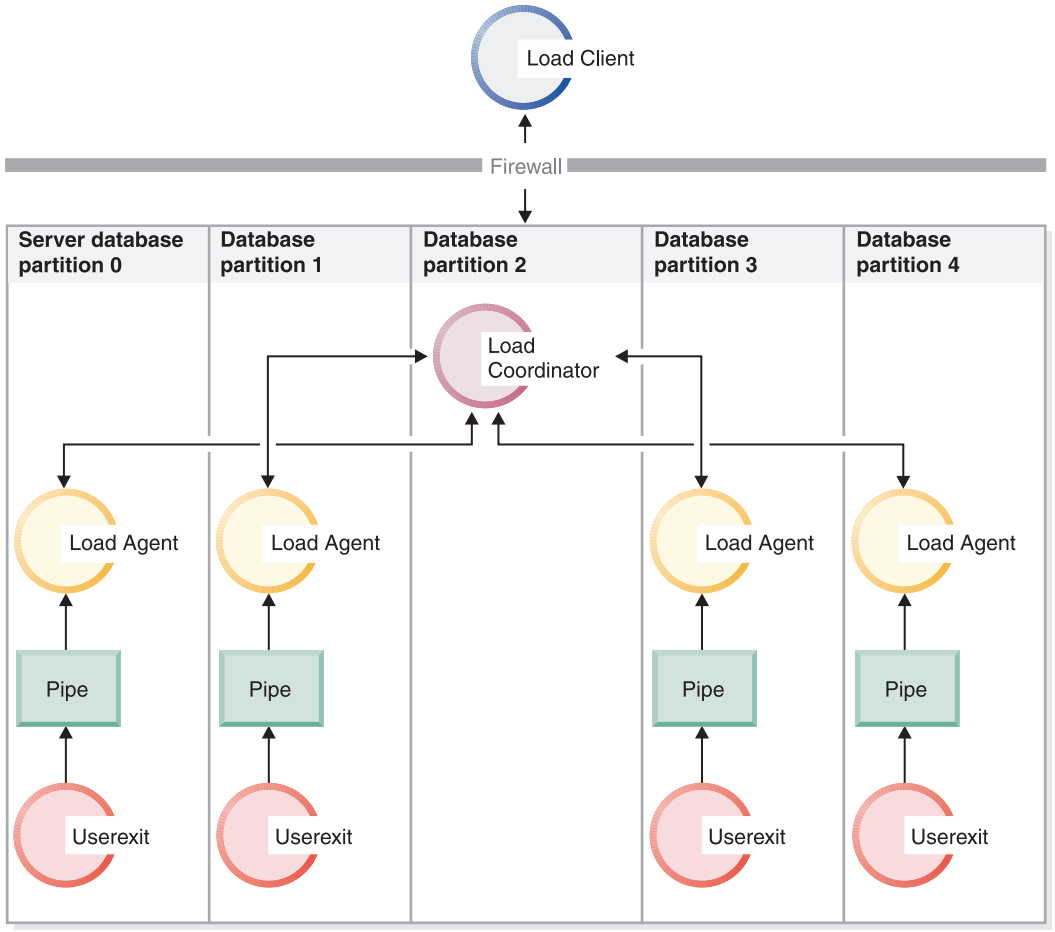


Figure 8. The various tasks performed when LOAD_ONLY or LOAD_ONLY_VERIFY_PART is specified.

Restrictions

- The LOAD_ONLY and LOAD_ONLY_VERIFY_PART partitioned-db-cfg mode options are not supported when the SOURCEUSEREXIT PARALLELIZE option is not specified.

Additional considerations for load

Parallelism and loading

The load utility takes advantage of a hardware configuration in which multiple processors or multiple storage devices are used, such as in a symmetric multiprocessor (SMP) environment.

There are several ways in which parallel processing of large amounts of data can take place using the load utility. One way is through the use of multiple storage devices, which allows for I/O parallelism during the load operation (see Figure 9 on page 153). Another way involves the use of multiple processors in an SMP environment, which allows for intra-partition parallelism (see Figure 10 on page 153). Both can be used together to provide even faster loading of data.

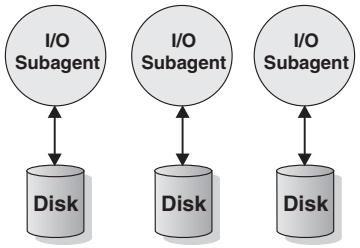


Figure 9. Taking Advantage of I/O Parallelism When Loading Data

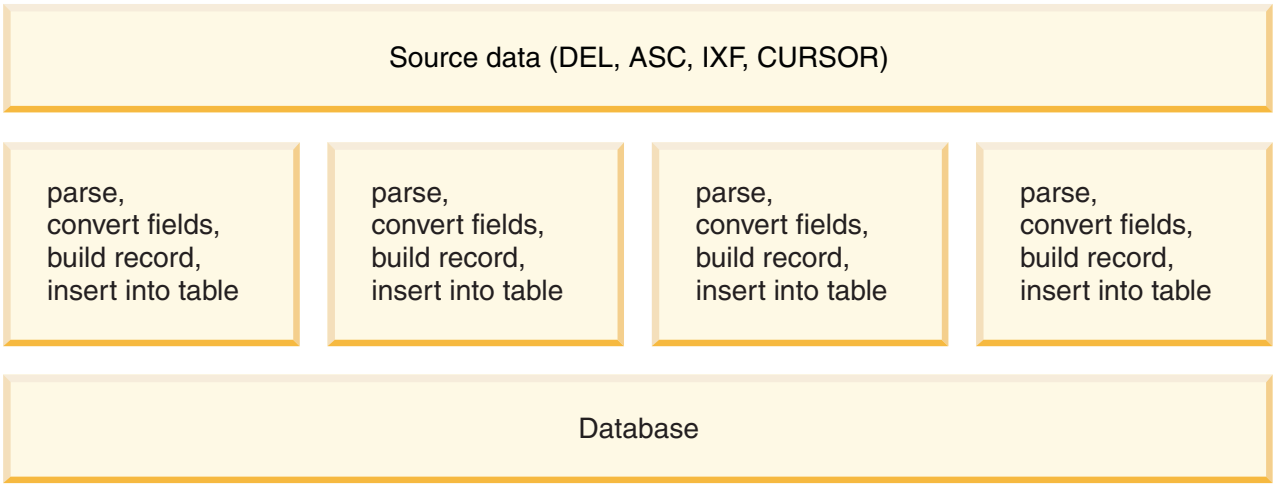


Figure 10. Taking Advantage of Intra-partition Parallelism When Loading Data

Index creation during load operations

Indexes are built during the build phase of a load operation. There are four indexing modes that can be specified in the LOAD command:

1. REBUILD. All indexes are rebuilt.
2. INCREMENTAL. Indexes are extended with new data.
3. AUTOSELECT. The load utility automatically decides between REBUILD or INCREMENTAL mode. AUTOSELECT is the default. If a load REPLACE operation is taking place, the REBUILD indexing mode is used. Otherwise, the indexing mode chosen is based on the ratio of the amount of existing data in the table to the amount of newly loaded data. If the ratio is sufficiently large, the INCREMENTAL indexing mode is chosen. Otherwise, the REBUILD indexing mode is chosen.
4. DEFERRED. The load utility does not attempt index creation if this mode is specified. Indexes are marked as needing a refresh, and a rebuild might be forced the first time they are accessed. The DEFERRED option is not allowed in any of the following situations:
   - If the ALLOW READ ACCESS option is specified (it does not maintain the indexes and index scanners require a valid index)
   - If any unique indexes are defined against the table
   - If XML data is being loaded (the XML Paths index is unique and is created by default whenever an XML column is added to a table)

Load operations that specify the ALLOW READ ACCESS option require special consideration in terms of space usage and logging depending on the type of

indexing mode chosen. When the ALLOW READ ACCESS option is specified, the load utility keeps indexes available for queries even while they are being rebuilt.

When a load operation in ALLOW READ ACCESS mode specifies the INDEXING MODE INCREMENTAL option, the load utility writes some log records that protect the integrity of the index tree. The number of log records written is a fraction of the number of inserted keys and is a number considerably less than would be needed by a similar SQL insert operation. A load operation in ALLOW NO ACCESS mode with the INDEXING MODE INCREMENTAL option specified writes only a small log record beyond the normal space allocation logs.

Note: This is only true if you did not specify COPY YES and have the *logindexrebuild* configuration parameter set to ON.

When a load operation in ALLOW READ ACCESS mode specifies the INDEXING MODE REBUILD option, new indexes are built as a *shadow* either in the same table space as the original index or in a system temporary table space. The original indexes remain intact and are available during the load operation and are only replaced by the new indexes at the end of the load operation while the table is exclusively locked. If the load operation fails and the transaction is rolled back, the original indexes remain intact.

By default, the shadow index is built in the same table space as the original index. Since both the original index and the new index are maintained simultaneously, there must be sufficient table space to hold both indexes at the same time. If the load operation is aborted, the extra space used to build the new index is released. If the load operation commits, the space used for the original index is released and the new index becomes the current index. When the new indexes are built in the same table space as the original indexes, replacing the original indexes takes place almost instantaneously.

If the indexes are built within an SMS table space, you can see index files in the table space directory with the `.IN1` suffix and the `.INX` suffix. These suffixes do not indicate which is the original index and which is the shadow index. However, if the indexes are built in a DMS table space, you cannot see the new shadow index.

Improving index creation performance

Building new indexes in a system temporary table space
The new index can be built in a system temporary table space to avoid running out of space in the original table space. The USE <tablespace-name> option allows the indexes to be rebuilt in a system temporary table space when using INDEXING MODE REBUILD and ALLOW READ ACCESS options. The system temporary table can be an SMS or a DMS table space, but the page size of the system temporary table space must match the page size of the original index table space.

The USE <tablespace-name> option is ignored if the load operation is not in ALLOW READ ACCESS mode, or if the indexing mode is incompatible. The USE <tablespace-name> option is only supported for the INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT options. If the INDEXING MODE AUTOSELECT option is specified and the load utility selects incremental maintenance of the indexes, the USE <tablespace-name> is ignored.

A load restart operation can use an alternate table space for building an index, even if the original load operation did not use an alternate table space. A load restart operation cannot be issued in ALLOW READ ACCESS mode if the original

load operation was not issued in ALLOW READ ACCESS mode. Load terminate operations do not rebuild indexes, so the USE <tablespace-name> is ignored.

During the build phase of the load operation, the indexes are built in the system temporary table space. Then, during the index copy phase, the index is copied from the system temporary table space to the original index table space. To make sure that there is sufficient space in the original index table space for the new index, space is allocated in the original table space during the build phase. So, if the load operation runs out of index space, it will do so during the build phase. If this happens, the original index is not lost.

The index copy phase occurs after the build and delete phases. Before the index copy phase begins, the table is locked exclusively. That is, it is unavailable for read access throughout the index copy phase. Since the index copy phase is a physical copy, the table might be unavailable for a significant amount of time.

Note: If either the system temporary table space or the index table space are DMS table spaces, the read from the system temporary table space can cause random I/O on the system temporary table space and can cause a delay. The write to the index table space is still optimized and the DISK_PARALLELISM values are used.

Considerations for large indexes
In order to improve performance when building large indexes during a load, it can be useful to tune the *sortheap* database configuration parameter. *sortheap* allocates the amount of memory dedicated to the sorting of index keys during a load operation. For example, to direct the load utility to use 4000 pages of main memory per index for key sorting, set *sortheap* to 4000 pages, disconnect all applications from the database, and then issue the LOAD command.

If an index is so large that it cannot be sorted in memory, a *sort spill*, or an overflow, occurs. That is, the data is divided among several "sort runs" and stored in a temporary table space that is merged later. Use the sort_overflows monitor element to determine whether or not a sort spill has occurred. If there is no way to avoid a sort spill by increasing the size of the *sortheap* parameter, ensure that the buffer pool for temporary table spaces be large enough to minimize the amount of disk I/O that spilling causes. Furthermore, to achieve I/O parallelism during the merging of sort runs, it is recommended that temporary table spaces be declared with multiple containers, each residing on a different disk device. If there is more than one index defined on a table, memory consumption increases proportionally because the load operation keeps all keys in memory.

Deferring index creation
Generally speaking, it is more efficient to allow indexes to be created during the load operation by specifying either REBUILD or INCREMENTAL mode than it is to have index creation deferred. As Figure 11 on page 156 indicates, tables are normally built in three steps: data loading, index building, and statistics collection. This causes multiple data I/O during the load operation, index creation (there can be several indexes for each table), and statistics collection (which causes I/O on the table data and on all of the indexes). A much faster alternative is to let the load utility complete all of these tasks in one pass through the data. It should be noted, however, that unique indexes reduce load performance if duplicates are encountered.

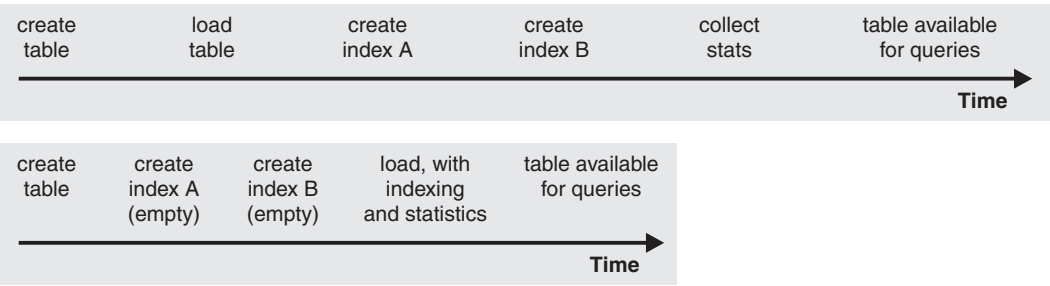


Figure 11. Increasing load performance through concurrent indexing and statistics collection. Tables are normally built in three steps: data loading, index building, and statistics collection. This causes multiple data I/O during the load operation, during index creation (there can be several indexes for each table), and during statistics collection (which causes I/O on the table data and on all of the indexes). A much faster alternative is to let the load utility complete all of these tasks in one pass through the data.

At certain times, deferring index creation and invoking the CREATE INDEX statement can improve performance. Sorting during index rebuild uses up to *sortheap* pages. If more space is required, TEMP buffer pool is used and (eventually) spilled to disk. If load spills, and thus decreases performance, it might be advisable to run LOAD with INDEXING MODE DEFERRED and re-create the index later. CREATE INDEX creates one index at a time, reducing memory usage while scanning the table multiple times to collect keys.

Another advantage of building indexes with a CREATE INDEX statement instead of concurrently with the load operation is that the CREATE INDEX statement can use multiple processes, or threads, to sort keys if INTRA PARALLEL is on. The actual building of the index is not executed in parallel.

Compression dictionary creation during load operations

Load INSERT and load REPLACE operations that meet certain criteria trigger automatic dictionary creation (ADC). Once enough data has been processed, ADC occurs if the load is performed on a table that has the COMPRESS attribute enabled and a compression dictionary is not present.

Data row compression uses a static dictionary-based compression algorithm to compress data. However, the dictionary must first exist in the table for compression to occur. During a load operation, the default behavior (indicated by the KEEPDICTIONARY option) is to either abide by the existing dictionary or, if a dictionary is not present, to generate one once a certain threshold (about 1 MB) of data has been scanned. The load utility uses the data that exists in the target table to build the dictionary, under the assumption that this data is representative of the kind of data that will be stored in that table. In cases where there is insufficient preexisting data in the target table, the load utility, once it has sampled enough input data, uses both the input data and preexisting data to build the dictionary.

When ADC occurs on range partitioned tables, each partition is treated like an individual table. There will not be any cross-partition dictionaries. ADC does not occur on partitions already containing dictionaries, and the dictionary generated for each partition is based only on the preexisting data (and, if necessary, the loaded data) in that partition.

Any load performed in the INSERT mode implicitly follows the KEEPDICTIONARY behavior. However, for load REPLACE operations, you have an additional choice: the RESETDICTIONARY option.

Load REPLACE using the KEEPDICTIONARY option
A load REPLACE that uses the KEEPDICTIONARY option keeps the existing dictionary and uses it to compress the loaded data, as long as the target table has the COMPRESS attribute enabled. If a dictionary does not exist, the load utility generates a new one (provided the data that is being loaded into the table surpasses the 1 MB threshold) for tables with the COMPRESS attribute enabled. Since the target table's data is being replaced, the load utility uses only the input data to build the dictionary. After the dictionary has been created, it is inserted into the table and the load operation continues.

Load REPLACE using the RESETDICTIONARY option
There are two key implications of using the RESETDICTIONARY option when loading into a table with the COMPRESS attribute on. First, ADC occurs as long as any amount of data will exist in the target table once the load REPLACE has completed. In other words, the new compression dictionary can be based on a single row of data. The other implication is that the existing dictionary is deleted but is not replaced (the target table will no longer have a compression dictionary) if any of the following situations are true:

- The operation is performed on a table with the COMPRESS attribute off
- Nothing was loaded (zero rows), in which case ADM5591W is printed to the notification log

Note: It you issue a load TERMINATE operation after a load REPLACE with the RESETDICTIONARY option, an existing compression dictionary will be deleted and not replaced.

Performance impact
ADC affects the performance of a load operation as a result of:

- The initial scan of table data
  For load INSERTs, all of the preexisting table data, not just the minimum threshold for ADC, is scanned prior to building the compression dictionary. Therefore, the time used for this scan increases with table size.
- The additional processing to build the compression dictionary
  The time actually used for building the dictionary is minimal. Moreover, once the dictionary has been built, ADC is turned off, by default.

Options for improving load performance

There are various command parameters that you can use to optimize load performance. There are also a number of file type modifiers unique to load which can, in some cases, significantly improve that utility's performance.

Command parameters

The load utility attempts to deliver the best performance possible by determining optimal values for DISK_PARALLELISM, CPU_PARALLELISM, and DATA BUFFER, if these parameters have not be specified by the user. Optimization is done based on the size and the free space available in the utility heap. Consider using the autonomic DISK_PARALLELISM and CPU_PARALLELISM settings before attempting to tune these parameters for your particular needs.

Following is information about the performance implications of various options available through the load utility:

ALLOW READ ACCESS
This option allows you to query a table while a load operation is in

progress. You can only view data that existed in the table prior to the load operation. If the INDEXING MODE INCREMENTAL option is also specified, and the load operation fails, the subsequent load terminate operation might have to correct inconsistencies in the index. This requires an index scan which involves considerable I/O. If the ALLOW READ ACCESS option is also specified for the load terminate operation, the buffer pool is used for I/O.

COPY YES or NO

Use this parameter to specify whether a copy of the input data is to be made during a load operation. COPY YES, which is only applicable when forward recovery is enabled, reduces load performance because all of the loading data is copied during the load operation. The increased I/O activity might increase the load time on an I/O-bound system. Specifying multiple devices or directories (on different disks) can offset some of the performance penalty resulting from this operation. COPY NO, which is only applicable when forward recovery is enabled, does not affect load performance. However, all table spaces related to the loaded table will be placed in a Backup Pending state, and those table spaces must be backed up before the table can be accessed.

CPU_PARALLELISM

Use this parameter to exploit intra-partition parallelism (if this is part of your machine's capability), and significantly improve load performance. The parameter specifies the number of processes or threads used by the load utility to parse, convert, and format data records. The maximum number allowed is 30. If there is insufficient memory to support the specified value, the utility adjusts the value. If this parameter is not specified, the load utility selects a default value that is based on the number of CPUs on the system.

Record order in the source data is preserved (see Figure 12) regardless of the value of this parameter, provided that:

- the `anyorder` file type modifier is not specified
- the PARTITIONING_DBPARTNUMS option (and more than one partition is to be used for partitioning) is not specified

If tables include either LOB or LONG VARCHAR data, CPU_PARALLELISM is set to 1. Parallelism is not supported in this case.

Although use of this parameter is not restricted to symmetric multiprocessor (SMP) hardware, you might not obtain any discernible performance benefit from using it in non-SMP environments.

Figure 12. Record Order in the Source Data is Preserved When Intra-partition Parallelism is Exploited During a Load Operation

DATA BUFFER

The DATA BUFFER parameter specifies the total amount of memory, in 4 KB units, allocated to the load utility as a buffer. It is recommended that this buffer be several *extents* in size. The data buffer is allocated from the utility heap. Depending on the amount of storage available on your system, you should consider allocating more memory for use by the DB2

utilities. The database configuration parameter *util_heap_sz* (utility heap size) can be modified accordingly. The default value for the *util_heap_sz* is 5 000 4 KB pages. Because load is only one of several utilities that use memory from the utility heap, it is recommended that no more than fifty percent of the pages defined by this parameter be available for the load utility, and that the utility heap be defined large enough.

DISK_PARALLELISM
The DISK_PARALLELISM parameter specifies the number of processes or threads used by the load utility to write data records to disk. Use this parameter to exploit available containers when loading data, and significantly improve load performance. The maximum number allowed is the greater of four times the CPU_PARALLELISM value (actually used by the load utility), or 50. By default, DISK_PARALLELISM is equal to the sum of the table space containers on all table spaces containing objects for the table being loaded, except where this value exceeds the maximum number allowed.

NONRECOVERABLE
If forward recovery is enabled, use this parameter if you do not need to be able to recover load transactions against a table upon rollforward. A NONRECOVERABLE load and a COPY NO load have identical performance. However, there is a significant difference in terms of potential data loss. A NONRECOVERABLE load marks a table as not rollforward recoverable while leaving the table fully accessible. This can create a problematic situation in which if you need to rollforward through the load operation, then the loaded data as well as all subsequent updates to the table will be lost. A COPY NO load places all dependent table spaces in the Backup Pending state which renders the table inaccessible until a backup is performed. Because you are forced to take a backup after that type of load, you will not risk losing the loaded data or subsequent updates to the table. That is to say, a COPY NO load is totally recoverable.

Note: When these load transactions are encountered during subsequent restore and rollforward recovery operations, the table is not updated, and is marked `invalid`. Further actions against this table are ignored. After the rollforward operation is complete, the table can only be dropped.

SAVECOUNT
Use this parameter to set an interval for the establishment of consistency points **during the load phase** of a load operation. The synchronization of activities performed to establish a consistency point takes time. If done too frequently, there is a noticeable reduction in load performance. If a very large number of rows is to be loaded, it is recommended that a large SAVECOUNT value be specified (for example, a value of 10 million in the case of a load operation involving 100 million records).

A load restart operation automatically continues from the last consistency point, provided that the load restart operation resumes from the load phase.

STATISTICS USE PROFILE
Collect statistics specified in table statistics profile. Use this parameter to collect data distribution and index statistics more efficiently than through invocation of the RUNSTATS utility following completion of the load operation, even though performance of the load operation itself decreases (particularly when DETAILED INDEXES ALL is specified).

For optimal performance, applications require the best data distribution and index statistics possible. Once the statistics are updated, applications can use new access paths to the table data based on the latest statistics. New access paths to a table can be created by rebinding the application packages using the BIND command. The table statistics profile is created by running the RUNSTATS command with the SET PROFILE options.

When loading data into large tables, it is recommended that a larger value for the *stat_heap_sz* (statistics heap size) database configuration parameter be specified.

USE <tablespace-name>
When an ALLOW READ ACCESS load is taking place and the indexing mode is REBUILD, this parameter allows an index to be rebuilt in a system temporary table space and copied back to the index table space during the index copy phase of a load operation.

By default, the fully rebuilt index (also known as the *shadow index*) is built in the same table space as the original index. This might cause resource problems as both the original and the shadow index reside in the same table space simultaneously. If the shadow index is built in the same table space as the original index, the original index is instantaneously replaced by the shadow. However, if the shadow index is built in a system temporary table space, the load operation requires an index copy phase which copies the index from a system temporary table space to the index table space. There is considerable I/O involved in the copy. If either of the table spaces is a DMS table space, the I/O on the system temporary table space might not be sequential. The values specified by the DISK_PARALLELISM option are respected during the index copy phase.

WARNINGCOUNT
Use this parameter to specify the number of warnings that can be returned by the utility before a load operation is forced to terminate. Set the WARNINGCOUNT parameter to a relatively low number if you are expecting only a few or no warnings. The load operation stops after the WARNINGCOUNT number is reached. This gives you the opportunity to correct problems before attempting to complete the load operation.

File type modifiers

ANYORDER
By default, the load utility preserves record order of source data. When load is operating under an SMP environment, synchronization between parallel processing is required to ensure that order is preserved.

In an SMP environment, specifying the `anyorder` file type modifier instructs the load utility to not preserve the order, which improves efficiency by avoiding the synchronization necessary to preserve that order. However, if the data to be loaded is presorted, `anyorder` might corrupt the presorted order, and the benefits of presorting are lost for subsequent queries.

Note: The `anyorder` file type modifier has no effect if CPU_PARALLELISM is 1, and it is not compatible with the SAVECOUNT option.

BINARYNUMERICS, **ZONEDDECIMAL** and **PACKEDDECIMAL**
For fixed length non-delimited ASCII (ASC) source data, representing numeric data in binary can result in improved performance when loading. If the `packeddecimal` file type modifier is specified, decimal data is interpreted by the load utility to be

in packed decimal format (two digits per byte). If the `zoneddecimal` file type modifier is specified, decimal data is interpreted by the load utility to be in zoned decimal format (one digit per byte). For all other numeric types, if the `binarynumerics` file type modifier is specified, data is interpreted by the load utility to be in binary format.

Note:

- When the `binarynumerics`, `packeddecimal`, or `zoneddecimal` file type modifiers are specified, numeric data is interpreted in big-endian (high byte first) format, regardless of platform.
- The `packeddecimal` and `zoneddecimal` file type modifiers are mutually exclusive.
- The `packeddecimal` and `zoneddecimal` file type modifiers only apply to the decimal target columns, and the binary data must match the target column definitions.
- The `reclen` file type modifier must be specified when the `binarynumerics`, `packeddecimal`, or `zoneddecimal` file type modifiers are specified.

FASTPARSE
Use with caution. In situations where the data being loaded is known to be valid, it can be unnecessary to have load perform the same amount of syntax checking as with more suspect data. In fact, decreasing the scope of this step can improve load's performance by about 10 or 20 percent. This can be done by using the `fastparse` file type modifier, which reduces the data checking that is performed on user-supplied column values from ASC and DEL files.

NOROWWARNINGS
During a load operation, warning messages about rejected rows are written to a specified file. However, if the load utility has to process a large volume of rejected, invalid or truncated records, it can adversely affect load's performance. In cases where many warnings are anticipated, it is useful to use the `norowwarnings` file type modifier to suppress the recording of these warnings.

PAGEFREESPACE, **INDEXFREESPACE**, and **TOTALFREESPACE**
As data is inserted and updated in tables over time, the need for table and index reorganization grows. One solution is to increase the amount of free space for tables and indexes using `pagefreespace`, `indexfreespace`, and `totalfreespace`. The first two modifiers, which take precedence over the PCTFREE value, specify the percentage of data and index pages that is to be left as free space, while `totalfreespace` specifies the percentage of the total number of pages that is to be appended to the table as free space.

Load features for maintaining referential integrity

Although the load utility is typically more efficient than the import utility, it requires a number of features to ensure the referential integrity of the information being loaded:

- **Table locks**, which provide concurrency control and prevent uncontrolled data access during a load operation
- **Table states** and **table space states**, which can either control access to data or elicit specific user actions
- **Load exception tables**, which ensure that rows of invalid data are not simply deleted without your knowledge

Checking for integrity violations following a load operation

Following a load operation, the loaded table might be in set integrity pending state in either READ or NO ACCESS mode if any of the following conditions exist:

- The table has table check constraints or referential integrity constraints defined on it.
- The table has generated columns and a V7 or earlier client was used to initiate the load operation.
- The table has descendent immediate materialized query tables or descendent immediate staging tables referencing it.
- The table is a staging table or a materialized query table.

The STATUS flag of the SYSCAT.TABLES entry corresponding to the loaded table indicates the set integrity pending state of the table. For the loaded table to be fully usable, the STATUS must have a value of N and the ACCESS MODE must have a value of F, indicating that the table is fully accessible and in normal state.

If the loaded table has descendent tables, the SET INTEGRITY PENDING CASCADE parameter can be specified to indicate whether or not the set integrity pending state of the loaded table should be immediately cascaded to the descendent tables.

If the loaded table has constraints as well as descendent foreign key tables, dependent materialized query tables and dependent staging tables, and if all of the tables are in normal state prior to the load operation, the following will result based on the load parameters specified:

INSERT, ALLOW READ ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
: The loaded table, its dependent materialized query tables and dependent staging tables are placed in set integrity pending state with read access.

INSERT, ALLOW READ ACCESS, and SET INTEGRITY PENDING CASCADE DEFERRED
: Only the loaded table is placed in set integrity pending with read access. Descendent foreign key tables, descendent materialized query tables and descendent staging tables remain in their original states.

INSERT, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
: The loaded table, its dependent materialized query tables and dependent staging tables are placed in set integrity pending state with no access.

INSERT or REPLACE, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE DEFERRED
: Only the loaded table is placed in set integrity pending state with no access. Descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables remain in their original states.

REPLACE, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
: The table and all its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables are placed in set integrity pending state with no access.

Note: Specifying the ALLOW READ ACCESS option in a load replace operation results in an error.

To remove the set integrity pending state, use the SET INTEGRITY statement. The SET INTEGRITY statement checks a table for constraints violations, and takes the table out of set integrity pending state. If all the load operations are performed in INSERT mode, the SET INTEGRITY statement can be used to incrementally process the constraints (that is, it checks only the appended portion of the table for constraints violations). For example:

```
db2 load from infile1.ixf of ixf insert into table1
db2 set integrity for table1 immediate checked
```

Only the appended portion of TABLE1 is checked for constraint violations. Checking only the appended portion for constraints violations is faster than checking the entire table, especially in the case of a large table with small amounts of appended data.

If a table is loaded with the SET INTEGRITY PENDING CASCADE DEFERRED option specified, and the SET INTEGRITY statement is used to check for integrity violations, the descendent tables are placed in set integrity pending state with no access. To take the tables out of this state, you must issue an explicit request.

If a table with dependent materialized query tables or dependent staging tables is loaded using the INSERT option, and the SET INTEGRITY statement is used to check for integrity violations, the table is taken out of set integrity pending state and placed in No Data Movement state. This is done to facilitate the subsequent incremental refreshes of the dependent materialized query tables and the incremental propagation of the dependent staging tables. In the No Data Movement state, operations that might cause the movement of rows within the table are not allowed.

You can override the No Data Movement state by specifying the FULL ACCESS option when you issue the SET INTEGRITY statement. The table is fully accessible, however a full re-computation of the dependent materialized query tables takes place in subsequent REFRESH TABLE statements and the dependent staging tables are forced into an incomplete state.

If the ALLOW READ ACCESS option is specified for a load operation, the table remains in read access state until the SET INTEGRITY statement is used to check for constraints violations. Applications can query the table for data that existed prior to the load operation once it has been committed, but will not be able to view the newly loaded data until the SET INTEGRITY statement is issued.

Several load operations can take place on a table before checking for constraints violations. If all of the load operations are completed in ALLOW READ ACCESS mode, only the data that existed in the table prior to the first load operation is available for queries.

One or more tables can be checked in a single invocation of this statement. If a dependent table is to be checked on its own, the parent table can not be in set integrity pending state. Otherwise, both the parent table and the dependent table must be checked at the same time. In the case of a referential integrity cycle, all the tables involved in the cycle must be included in a single invocation of the SET INTEGRITY statement. It might be convenient to check the parent table for constraints violations while a dependent table is being loaded. This can only occur if the two tables are not in the same table space.

When issuing the SET INTEGRITY statement, you can specify the INCREMENTAL option to explicitly request incremental processing. In most cases, this option is not

needed, because the DB2 database selects incremental processing. If incremental processing is not possible, full processing is used automatically. When the INCREMENTAL option is specified, but incremental processing is not possible, an error is returned if:

- New constraints are added to the table while it is in set integrity pending state.
- A load replace operation takes place, or the NOT LOGGED INITIALLY WITH EMPTY TABLE option is activated, after the last integrity check on the table.
- A parent table is load replaced or checked for integrity non-incrementally.
- The table is in set integrity pending state before migration. Full processing is required the first time the table is checked for integrity after migration.
- The table space containing the table or its parent is rolled forward to a point in time and the table and its parent reside in different table spaces.

If a table has one or more `W` values in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and if the NOT INCREMENTAL option is not specified in the SET INTEGRITY statement, the table is incrementally processed and the CONST_CHECKED column of SYSCAT.TABLES is marked as `U` to indicate that not all data has been verified by the system.

The SET INTEGRITY statement does not activate any DELETE triggers as a result of deleting rows that violate constraints, but once the table is removed from set integrity pending state, triggers are active. Thus, if you correct data and insert rows from the exception table into the loaded table, any INSERT triggers defined on the table are activated. The implications of this should be considered. One option is to drop the INSERT trigger, insert rows from the exception table, and then recreate the INSERT trigger.

Checking for constraint violations using SET INTEGRITY

Typically, you need to manually perform integrity processing for a table in three situations: After loading data into a table; when altering a table by adding constraints on the table; and when altering a table to add a generated column.

- To turn on constraint checking for a table and performing integrity processing on the table, you need one of the following:
  - SYSADM or DBADM authority
  - CONTROL privileges on the tables being checked, and if exceptions are being posted to one or more tables, INSERT privilege on the exception tables
  - CONTROL privilege on all descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables that will implicitly be placed in the Set Integrity Pending state by the statement
  - LOAD authority, and if exceptions are being posted to one or more tables:
    - SELECT and DELETE privilege on each table being checked
    - INSERT privilege on the exception tables
- To turn on constraint checking for a table without performing integrity processing on the table, you need one of the following:
  - SYSADM or DBADM authority
  - CONTROL privileges on the tables being checked
  - CONTROL privilege on each descendent foreign key table, descendent immediate materialized query table, and descendent immediate staging table that will implicitly be placed in the Set Integrity Pending state by the statement

- To turn off constraint checking, immediate refreshing, or immediate propagation for tables, you need one of the following:
  - SYSADM or DBADM authority
  - CONTROL privilege on the table, and on all descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables that will have their integrity checking turned off by the statement
  - LOAD authority

The load operation causes a table to be put into Set Integrity Pending state automatically if the table has constraints defined on it or if it has dependent foreign key tables, dependent materialized query tables, or dependent staging tables. When the load operation is completed, you can verify the integrity of the loaded data and you can turn on constraint checking for the table. If the table has dependent foreign key tables, dependent materialized query tables, or dependent staging tables, they will be automatically put into Set Integrity Pending state. You will need to use the Set Integrity window to perform separate integrity processing on each of these tables.

If you are altering a table by adding a foreign key, a check constraint or a generated column, you need to turn off constraint checking before you alter the table. After you add the constraint, you need to check the existing data for violations to the newly added constraint and you need to turn constraint checking back on. In addition, if you are loading data into the table, you cannot activate constraint checking on the table until you complete loading data into it. If you are importing data into the table, you should activate constraint checking on the table before you import data into it.

Constraints checking refers to checking for constraints violations, foreign key violations, and generated columns violations. Integrity processing refers to populating identity and generated columns, refreshing materialized query tables, and propagating to staging tables, in addition to performing constraints checking.

Normally, referential integrity and check constraints on a table are automatically enforced, materialized query tables are automatically refreshed immediately, and staging tables are automatically propagated. In some situations, you might need to manually change this behavior.

To check for constraint violations using the Control Center:

1. Open the Set Integrity window: From the Control Center, expand the object tree until you find the **Tables** folder. Click on the **Tables** folder. Any existing tables are displayed in the pane on the right side of the window. Right-click the table you want and select **Set Integrity** from the pop-up menu. The Set Integrity window opens.
2. Review the Current Integrity Status of the table you are working with.
3. To turn on constraint checking for a table and not check the table data:
   a. Select the **Immediate and unchecked** radio button.
   b. Specify the type of integrity processing that you are turning on.
   c. Select the **Full Access** radio button to immediately perform data movement operations against the table (such as reorganize or redistribute). However, note that subsequent refreshes of dependent materialized query tables will take longer. If the table has an associated materialized query table, it is recommended that you do not select this radio button in order to reduce the time needed to refresh the materialized query table.

4. To turn on constraint checking for a table and check the existing table data:
   a. Select the **Immediate and checked** radio button.
   b. Select which type of integrity processing that you want to perform. If the **Current integrity status** shows that the constraints checked value for the materialized query table is incomplete, you cannot incrementally refresh the materialized query table.
   c. Optional: If you want identity or generated columns to be populated during integrity processing, select the **Force generated** check box.
   d. If the table is not a staging table, make sure that the **Prune** check box is unchecked.
   e. Select the **Full Access** radio button to immediately perform data movement operations against the table.
   f. Optional: Specify an exception table. Any row that is in violation of a referential or check constraint will be deleted from your table and copied to the exception table. If you do not specify an exception table, when a constraint is violated, only the first violation detected is returned to you and the table is left in the Set Integrity Pending state.
5. To turn off constraint checking, immediate refreshing, or immediate propagation for a table:
   a. Select the **Off** radio button. The table will be put in Set Integrity Pending state.
   b. Use the **Cascade** option to specify whether you want to cascade immediately or defer cascading. If you are cascading immediately, use the **Materialized Query Tables**, **Foreign Key Tables**, and **Staging Tables** check boxes to indicate the tables to which you want to cascade.

      Note: If you turn off constraint checking for a parent table and specify that you want to cascade the changes to foreign key tables, the foreign key constraints of all of its descendent foreign key tables are also turned off. If you turn off constraint checking for a underlying table and specify that you want to cascade the check pending state to materialized query tables, the refresh immediate properties of all its dependent materialized query tables are also turned off. If you turn off constraint checking for a underlying table and specify that you want to cascade the Set Integrity Pending state to staging tables the propagate immediate properties of all its dependent staging tables are also turned off.

To check for constraint violations using the command line, use the SET INTEGRITY statement.

Troubleshooting tip

Symptom

You receive the following error message when you try to turn on constraints checking, immediate refresh, or immediate propagation for a table:

DB2 Message

Cannot check a dependent table TABLE1 using the SET INTEGRITY statement while the parent table or underlying table TABLE2 is in the Set Integrity Pending state or if it will be put into the Set Integrity Pending state by the SET INTEGRITY statement.

Where TABLE1 is the table for which you are trying to turn on constraints checking, immediate refresh, or immediate propagation and it is dependent on TABLE2.

Possible cause

Constraint checking, immediate refresh, or immediate propagation cannot be turned on for a table that has a parent or underlying table in Set Integrity Pending.

Action

Bring the parent or underlying table out of Set Integrity Pending by turning on constraint checking for the table. Begin with the table identified as the parent or underlying table in the DB2 message. If that table is dependent on another table, you need to turn on constraint checking in a top-down approach from the table at the top of the dependency chain.

Attention: If the selected table has a cyclical referential constraint relationship with one or more tables, you cannot use the Set Integrity window to turn on constraint checking. In this case, you must use the Command Editor to issue the SQL SET INTEGRITY command.

Table locking during load operations

In most cases, the load utility uses table level locking to restrict access to tables. The level of locking depends on the stage of the load operation and whether it was specified to allow read access.

A load operation in ALLOW NO ACCESS mode uses a super exclusive lock (Z-lock) on the table for the duration of the load.

Before a load operation in ALLOW READ ACCESS mode begins, the load utility waits for all applications that began before the load operation to release their locks on the target table. At the beginning of the load operation, the load utility acquires an update lock (U-lock) on the table. It holds this lock until the data is being committed. When the load utility acquires the U-lock on the table, it waits for all applications that hold locks on the table prior to the start of the load operation to release them, even if they have compatible locks. This is achieved by temporarily upgrading the U-lock to a Z-lock which does not conflict with new table lock requests on the target table as long as the requested locks are compatible with the load operation's U-lock. When data is being committed, the load utility upgrades the lock to a Z-lock, so there can be some delay in commit time while the load utility waits for applications with conflicting locks to finish.

Note: The load operation can time out while it waits for the applications to release their locks on the table prior to loading. However, the load operation does not time out while waiting for the Z-lock needed to commit the data.

Applications with conflicting locks

Use the LOCK WITH FORCE option of the LOAD command to force off applications holding conflicting locks on a target table so that the load operation can proceed. Before a load operation in ALLOW READ ACCESS mode can proceed, applications holding the following locks are forced off:

- Table locks that conflict with a table update lock (for example, import or insert).
- All table locks that exist at the commit phase of the load operation.

Applications holding conflicting locks on the system catalog tables are not forced off by the load utility. If an application is forced off the system by the load utility, the application loses its database connection, and an error is returned (SQL1224N).

When you specify the COPY NO option for a load operation on a recoverable database, all objects in the target table space are locked in share mode before the table space is placed in the Backup Pending state. This occurs regardless of the access mode. If you specify the LOCK WITH FORCE option, all applications holding locks on objects in the table space that conflict with a share lock are forced off.

Read access load operations

The load utility provides two options that control the amount of access other applications have to a table being loaded. The ALLOW NO ACCESS option locks the table exclusively and allows no access to the table data while the table is being loaded.

The ALLOW NO ACCESS option is the default behavior. The ALLOW READ ACCESS option prevents all write access to the table by other applications, but allows read access to preexisting data. This section deals with the ALLOW READ ACCESS option.

Table data and index data that exist prior to the start of a load operation are visible to queries while the load operation is in progress. Consider the following example:

1. Create a table with one integer column:

```
create table ED (ed int)
```

2. Load three rows:

```
load from File1 of del insert into ED
...
Number of rows read         = 3
Number of rows skipped      = 0
Number of rows loaded       = 3
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 3
```

3. Query the table:

```
select * from ED

ED
-----------
          1
          2
          3

  3 record(s) selected.
```

4. Perform a load operation with the ALLOW READ ACCESS option specified and load two more rows of data:

```
load from File2 of del insert into ED allow read access
```

5. At the same time, on another connection query the table while the load operation is in progress:

```
select * from ED

ED
-----------
          1
```

```
          2
          3

  3 record(s) selected.
```

6. Wait for the load operation to finish and then query the table:

```
select * from ED

ED
-----------
          1
          2
          3
          4
          5

  5 record(s) selected.
```

The ALLOW READ ACCESS option is very useful when loading large amounts of data because it gives users access to table data at all times, even when the load operation is in progress or after a load operation has failed. The behavior of a load operation in ALLOW READ ACCESS mode is independent of the isolation level of the application. That is, readers with any isolation level can always read the preexisting data, but they are not be able to read the newly loaded data until the load operation has finished.

Read access is provided throughout the load operation except for two instances: at the beginning and at the end of the operation.

Firstly, the load operation acquires a special Z-lock for a short duration of time near the end of its setup phase. If an application holds an incompatible lock on the table prior to the load operation requesting this special Z-lock, then the load operation waits a finite amount of time for this incompatible lock to be released before timing out and failing. The amount of time is determined by the *locktimeout* database configuration parameter. If the LOCK WITH FORCE option is specified then the load operation forces other applications off to avoid timing out. The load operation acquires the special Z-lock, commits the phase, releases the lock, and then continues onto the load phase. Any application that requests a lock on the table for reading after the start of the load operation in ALLOW READ ACCESS mode is granted the lock, and it does not conflict with this special Z-lock. New applications attempting to read existing data from the target table are able to do so.

Secondly, before data is committed at the end of the load operation, the load utility acquires an exclusive lock (Z-lock) on the table. The load utility waits until all applications that hold locks on the table release them. This can cause a delay before the data is committed. The LOCK WITH FORCE option is used to force off conflicting applications, and allow the load operation to proceed without having to wait. Usually, a load operation in ALLOW READ ACCESS mode acquires an exclusive lock for a short amount of time; however, if the USE <tablespace-name> option is specified, the exclusive lock lasts for the entire period of the index copy phase.

When the load utility is running against a table defined on multiple database partitions, the load process model executes on each individual database partition, meaning that locks are acquired and released independently of other db-partitions. Thus, if a query or other operation is executed concurrently and is competing for the same locks, there is a chance for deadlocks. For example, suppose that operation A is granted a table lock on db-partition 0 and the load operation is

granted a table lock on db-partition 1. A deadlock can occur because operation A is waiting to be granted a table lock on db-partition 1, while the load operation is waiting for a table lock on db-partition 0. In this case, the deadlock detector will arbitrarily roll back one of the operations.

Note:

1. If a load operation is interrupted or fails, it remains at the same access level that was specified when the load operation was issued. That is, if a load operation in ALLOW NO ACCESS mode fails, the table data is inaccessible until a load terminate or a load restart is issued. If a load operation in ALLOW READ ACCESS mode aborts, the preexisting table data is still accessible for read access.
2. If the ALLOW READ ACCESS option was specified for an interrupted or failed load operation, it can also be specified for the load restart or load terminate operation. However, if the interrupted or failed load operation specified the ALLOW NO ACCESS option, the ALLOW READ ACCESS option cannot be specified for the load restart or load terminate operation.

The ALLOW READ ACCESS option is not supported if:

- The REPLACE option is specified. Since a load replace operation truncates the existing table data before loading the new data, there is no preexisting data to query until after the load operation is complete.
- The indexes have been marked invalid and are waiting to be rebuilt. Indexes can be marked invalid in some rollforward scenarios or through the use of the db2dart command.
- The INDEXING MODE DEFERRED option is specified. This mode marks the indexes as requiring a rebuild.
- An ALLOW NO ACCESS load operation is being restarted or terminated. Until it is brought fully online, a load operation in ALLOW READ ACCESS mode cannot take place on the table.
- A load operation is taking place to a table that is in Set Integrity Pending No Access state. This is also the case for multiple load operations on tables with constraints. A table is not brought online until the SET INTEGRITY statement is issued.

Generally, if table data is taken offline, read access is not available during a load operation until the table data is back online.

Table space states during and after load operations

The load utility uses table space states to preserve database consistency during a load operation. These states work by controlling access to data or eliciting user actions.

The load utility does not quiesce (put persistent locks on) the table spaces involved in the load operation and uses table space states only for load operations for which you specify the COPY NO parameter.

You can check table space states by using the LIST TABLESPACES command. Table spaces can be in multiple states simultaneously. The states returned by LIST TABLESPACES are as follows:

Normal
The Normal state is the initial state of a table space after it is created, indicating that no (abnormal) states currently affect it.

Load in Progress
The Load in Progress state indicates that there is a load in progress on the table space. This state prevents the backup of dependent tables during the load. The table space state is distinct from the Load in Progress table state (which is used in all load operations) because the load utility places table spaces in the Load in Progress state only when you specify the **COPY NO** parameter for a recoverable database. The table spaces remain in this state for the duration of the load operation.

Backup Pending
If you perform a load operation for a recoverable database and specify the **COPY NO** parameter, table spaces are placed in the Backup Pending table space state after the first commit. You cannot update a table space in the Backup Pending state. You can remove the table space from the Backup Pending state only by backing up the table space. Even if you cancel the load operation, the table space remains in the Backup Pending state because the table space state is changed at the beginning of the load operation and cannot be rolled back.

Restore Pending
If you perform a successful load operation with the **COPY NO** option, restore the database, and then rollforward through that operation, the associated table spaces are placed in the Restore Pending state. To remove the table spaces from the Restore Pending state, you must perform a restore operation.

Example of a table space state

If you load an input file (staffdata.del) into a table NEWSTAFF, as follows:

```
update db cfg for sample using logretain recovery;
backup db sample;
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff copy no;
connect reset;
```

and you open another session and issue the following commands,

```
connect to sample;
list tablespaces;
connect reset;
```

USERSPACE1 (the default table space for the sample database) is in the Load in Progress state and, after the first commit, the Backup Pending state as well. After the load operation finishes, the LIST TABLESPACES command reveals that USERSPACE1 is now in the Backup Pending state:

```
Tablespace ID                        = 2
Name                                 = USERSPACE1
Type                                 = Database managed space
Contents                             = All permanent data. Large table space.
State                                = 0x0020
  Detailed explanation:
  Backup pending
```

Table states during and after load operations

The load utility uses table states to preserve database consistency during a load operation. These states work by controlling access to data or eliciting user actions.

To determine the state of a table, issue the LOAD QUERY command, which also checks the status of a load operation. Tables can be in a number of states simultaneously. The states returned by LOAD QUERY are as follows:

Normal State
The Normal state is the initial state of a table after it is created, indicating that no (abnormal) states currently affect the table.

Read Access Only
If you specify the ALLOW READ ACCESS option, the table is in the Read Access Only state. The data in the table that existed prior to the invocation of the load command is available in read-only mode during the load operation. If you specify the ALLOW READ ACCESS option and the load operation fails, the data that existed in the table prior to the load operation continues to be available in read-only mode after the failure.

Load in Progress
The Load in Progress table state indicates that there is a load in progress on the table. The load utility removes this transient state after the load is successfully completed. However, if the load operation fails or is interrupted, the table state will change to Load Pending.

Redistribute in Progress
The Redistribute in Progress table state indicates that there is a redistribute in progress on the table. The redistribute utility removes this transient state after it has successfully completed processing the table. However, if the redistribute operation fails or is interrupted, the table state will change to Redistribute Pending.

Load Pending
The Load Pending table state indicates that a load operation failed or was interrupted. You can take one of the following steps to remove the Load Pending state:

- Address the cause of the failure. For example, if the load utility ran out of disk space, add containers to the table space. Then, restart the load operation.
- Terminate the load operation.
- Run a load REPLACE operation against the same table on which the load operation failed.
- Recover table spaces for the loading table by using the RESTORE DATABASE command with the most recent table space or database backup, then carry out further recovery actions.

Redistribute Pending
The Redistribute Pending table state indicates that a redistribute operation failed or was interrupted. You can perform a REDISTRIBUTE CONTINUE or REDISTRIBUTE ABORT operation to remove the Redistribute Pending state.

Not Load Restartable
In the Not Load Restartable state, a table is partially loaded and does not allow a load restart operation. There are two situations in which a table is placed in the Not Load Restartable state:

- If you perform a rollforward operation after a failed load operation that you could not successfully restart or terminate
- If you perform a restore operation from an online backup that you took while the table was in the Load in Progress or Load Pending state

The table is also in the Load Pending state. To remove the table from the Not Load Restartable state, issue the LOAD TERMINATE or the LOAD REPLACE command.

Set Integrity Pending
The Set Integrity Pending state indicates that the loaded table has constraints which have not yet been verified. The load utility places a table in this state when it begins a load operation on a table with constraints. Use the SET INTEGRITY statement to take the table out of Set Integrity Pending state.

Type-1 Indexes
The Type-1 Indexes state indicates that the table currently uses type-1 indexes, which have been deprecated. The indexes can be converted to type-2 using the CONVERT option when using the REORG utility on the indexes.

Unavailable
Rolling forward through an unrecoverable load operation places a table in the Unavailable state. In this state, the table is unavailable; you must drop it or restore it from a backup.

Example of a table in multiple states

If you load an input file (staffdata.del) with a substantial amount of data into a table NEWSTAFF, as follows:

```
connect to sample;
create table newstaff like staff;
load from staffdata.del of del insert into newstaff allow read access;
connect reset;
```

and you open another session and issue the following commands,

```
connect to sample;
load query table newstaff;
connect reset;
```

the LOAD QUERY command reveals that the NEWSTAFF table is in the Read Access Only and Load in Progress table states:

```
Tablestate:
 Load in Progress
 Read Access Only
```

Load exception tables

A load exception table is a consolidated report of all of the rows that violated unique index rules, range constraints, and security policies during a load operation. You specify a load exception table by using the FOR EXCEPTION clause of the LOAD command.

Restriction: An exception table can not contain an identity column or any other type of generated column. If an identity column is present in the primary table, the corresponding column in the exception table should only contain the column's type, length, and nullability attributes. In addition, the exception table cannot be partitioned or have a unique index.

The exception table used with the load utility is identical to the exception tables used by the SET INTEGRITY statement. It is a user-created table that reflects the definition of the table being loaded and includes some additional columns.

You can assign a load exception table to the table space where the table being loaded resides or to another table space. In either case, assign the load exception table and the table being loaded to the same database partition group, and ensure that both tables use the same distribution key.

When to use an exception table
Use an exception table when loading data that has a unique index and could have duplicate records. If you do not specify an exception table and duplicate records are found, the load operation continues, and only a warning message is issued about the deleted duplicate records. The duplicate records are not logged.

After the load operation is completed, you can use information in the exception table to correct data that is in error. You can then insert the corrected data into the table.

Rows are appended to existing information in the exception table. Because there is no checking done to ensure that the table is empty, new information is simply added to the invalid rows from previous load operations. If you want only the invalid rows from the current load operation, you can remove the existing rows before invoking the utility. Alternatively, when you define a load operation, you can specify that the exception table record the time when a violation is discovered and the name of the constraint violated.

Because each deletion event is logged, the log could fill up during the delete phase of the load if there are a large number of records that violate a uniqueness condition.

Any rows rejected because of invalid data before the building of an index are not inserted into the exception table.

Failed or incomplete loads

Restarting an interrupted load operation

If a failure or interruption occurs during a load operation, you can use the load utility to terminate the operation, reload the table, or restart the load operation.

If the load utility does not even start because of a user error such as a nonexistent data file or invalid column names, the operation terminates and leaves the target table in a normal state.

When the load operation begins, the target table is placed in the Load in Progress table state. In the event of a failure, the table state will change to Load Pending. To remove the table from this state, you can issue a LOAD TERMINATE to roll back the operation, issue a LOAD REPLACE to reload the entire table, or issue a LOAD RESTART.

Typically, restarting the load operation is the best choice in this situation. It saves time because the load utility restarts the load operation from the last successfully reached point in its progress, rather than from the beginning of the operation. Where exactly the operation restarts from depends upon the parameters specified in the original command. If the SAVECOUNT option was specified, and the previous load operation failed in the load phase, the load operation restarts at the last consistency point it reached. Otherwise, the load operation restarts at the beginning of the last phase successfully reached (the load, build, or delete phase).

If you are loading XML documents, the behavior is slightly different. Because the SAVECOUNT option is not supported with loading XML data, load operations that fail during the load phase restart from the beginning of the operation. Just as with other data types, if the load fails during the build phase, indexes are built in REBUILD mode, so the table is scanned to pick up all index keys from each row; however, each XML document must also be scanned to pick up the index keys. This process of scanning XML documents for keys requires them to be reparsed, which is an expensive operation. Furthermore, the internal XML indexes, such as the regions and paths indexes, need to be rebuilt first, which also requires a scan of the XDA object.

Once you have fixed the situation that caused the load operation to fail, reissue the load command. Ensure that you specify exactly the same parameters as in the original command, so that the load utility can find the necessary temporary files. An exception to this is if you want to disallow read access. A load operation that specified the ALLOW READ ACCESS option can also be restarted as an ALLOW NO ACCESS option.

Note: Do not delete or modify any temporary files created by the load utility.

If the load operation resulting from the following command fails,

```
LOAD FROM file_name OF file_type
SAVECOUNT n
MESSAGES message_file
load_method
INTO target_tablename
```

you would restart it by replacing the specified load method (load_method) with the RESTART method:

```
LOAD FROM file_name OF file_type
SAVECOUNT n
MESSAGES message_file
RESTART
INTO target_tablename
```

Failed loads that cannot be restarted

You cannot restart failed or interrupted load operations if the table involved in the operation is in the Not Load Restartable table state. Tables are put in that state for the following reasons:

- A rollforward operation is performed after a failed load operation that has not been successfully restarted or terminated
- A restore operation is performed from an online backup that was taken while the table was in the Load in Progress or Load Pending table state

You should issue either a LOAD TERMINATE or a LOAD REPLACE command.

Restarting or terminating an ALLOW READ ACCESS load operation

An interrupted or cancelled load operation that specifies the ALLOW READ ACCESS option can also be restarted or terminated using the ALLOW READ ACCESS option. Using the ALLOW READ ACCESS option allows other applications to query the table data while the terminate or restart operation is in progress. As with a load operation in ALLOW READ ACCESS mode, the table is locked exclusively prior to the data being committed.

If the index object is unavailable or marked invalid, a load restart or terminate operation in ALLOW READ ACCESS mode is not permitted.

If the original load operation is interrupted or cancelled in the index copy phase, a restart operation in the ALLOW READ ACCESS mode is not permitted because the index might be corrupted.

If a load operation in ALLOW READ ACCESS mode is interrupted or cancelled in the load phase, it restarts in the load phase. If it is interrupted or cancelled in any phase other than the load phase, it restarts in the build phase. If the original load operation is in ALLOW NO ACCESS mode, a restart operation occurs in the delete phase if the original load operation reaches that point and the index is valid. If the index is marked invalid, the load utility restarts the load operation from the build phase.

Note: All load restart operations choose the REBUILD indexing mode even if the INDEXING MODE INCREMENTAL option is specified.

Issuing a LOAD TERMINATE command generally causes the interrupted or cancelled load operation to be rolled back with minimal delay. However, when issuing a LOAD TERMINATE command for a load operation where ALLOW READ ACCESS and INDEXING MODE INCREMENTAL are specified, there might be a delay while the load utility scans the indexes and corrects any inconsistencies. The length of this delay depends on the size of the indexes and occurs whether or not the ALLOW READ ACCESS option is specified for the load terminate operation. The delay does not occur if the original load operation failed prior to the build phase.

Note: The delay resulting from corrections to inconsistencies in the index is considerably less than the delay caused by marking the indexes as invalid and rebuilding them.

A load restart operation cannot be undertaken on a table that is in the Not Load Restartable table state. A table can be placed in the Not Load Restartable table state during a rollforward operation. This can occur if you roll forward to a point in time that is prior to the end of a load operation, or if you roll forward through an interrupted or cancelled load operation but do not roll forward to the end of the load terminate or load restart operation.

Recovering data with the load copy location file

The **DB2LOADREC** registry variable is used to identify the file with the load copy location information. This file is used during rollforward recovery to locate the load copy.

DB2LOADREC has information about:

- Media type
- Number of media devices to be used
- Location of the load copy generated during a table load operation
- File name of the load copy, if applicable

If the location file does not exist, or no matching entry is found in the file, the information from the log record is used.

The information in the file might be overwritten before rollforward recovery takes place.

Note:

1. In a multi-partition database, the **DB2LOADREC** registry variable must be set for all the database partition servers using the db2set command.
2. In a multi-partition database, the load copy file must exist at each database partition server, and the file name (including the path) must be the same.
3. If an entry in the file identified by the **DB2LOADREC** registry variable is not valid, the old load copy location file is used to provide information to replace the invalid entry.

The following information is provided in the location file. The first five parameters must have valid values, and are used to identify the load copy. The entire structure is repeated for each load copy recorded. For example:

```
TIMestamp      19950725182542         * Time stamp generated at load time
DBPartition    0                      * DB Partition number (OPTIONAL)
SCHema         PAYROLL                * Schema of table loaded
TABlename      EMPLOYEES              * Table name
DATabasename   DBT                    * Database name
DB2instance    toronto                * DB2INSTANCE
BUFfernumber   NULL                   * Number of buffers to be used for
                                        recovery
SESsionnumber  NULL                   * Number of sessions to be used for
                                        recovery
TYPeofmedia    L                      * Type of media - L for local device
                                                       A for TSM
                                                       O for other vendors
LOCationnumber 3                      * Number of locations
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
DBP            18
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            toronto
BUF            NULL
SES            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            toronto
BUF            NULL
SES            NULL
TYP            O
SHRlib         /@sys/lib/backup_vendor.a
```

Note:

1. The first three characters in each keyword are significant. All keywords are required in the specified order. Blank lines are not accepted.
2. The time stamp is in the form *yyyymmddhhmmss*.
3. All fields are mandatory, except for BUF and SES (which can be NULL), and DBP (which can be missing from the list). If SES is NULL, the value specified by the *dft_loadrec_ses* configuration parameter is used. If BUF is NULL, the default value is SES+2.

4. If even one of the entries in the location file is invalid, the previous load copy location file is used to provide those values.
5. The media type can be local device (`L` for tape, disk or diskettes), TSM (`A`), or other vendor (`O`). If the type is `L`, the number of locations, followed by the location entries, is required. If the type is `A`, no further input is required. If the type is `O`, the shared library name is required.
6. The SHRlib parameter points to a library that has a function to store the load copy data.
7. If you invoke a load operation, specifying the COPY NO or the NONRECOVERABLE option, and do not take a backup copy of the database or affected table spaces after the operation completes, you cannot restore the database or table spaces to a point in time that follows the load operation. That is, you cannot use rollforward recovery to recreate the database or table spaces to the state they were in following the load operation. You can only restore the database or table spaces to a point in time that precedes the load operation.

If you want to use a particular load copy, you can use the recovery history file for the database to determine the time stamp for that specific load operation. In a multi-partition database, the recovery history file is local to each database partition.

Load dump file

Specifying the `dumpfile` file type modifier tells the load utility the name and the location of the exception file to which rejected rows are written.

When running in a partitioned database environment, rows can be rejected either by the partitioning subagents or by the loading subagents. Because of this, the dump file name is given an extension that identifies the subagent type, as well as the database partition number where the exceptions were generated. For example, if you specified the following dump file value:

```
dumpfile = "/u/usrname/dumpit"
```

Then rows that are rejected by the load subagent on database partition five will be stored in a file named `/u/usrname/dumpit.load.005`, rows that are rejected by the load Subagent on database partition two will be stored in a file named `/u/usrname/dumpit.load.002`, and rows that are rejected by the partitioning subagent on database partition two will be stored in a file named `/u/usrname/dumpit.part.002`, and so on.

For rows rejected by the load subagent, if the row is less than 32 768 bytes in length, the record is copied to the dump file in its entirety; if it is longer, a row fragment (including the final bytes of the record) is written to the file.

For rows rejected by the partitioning subagent, the entire row is copied to the dump file regardless of the record size.

Load temporary files

DB2 creates temporary binary files during load processing. These files are used for load crash recovery, load terminate operations, warning and error messages, and runtime control data.

Load temporary files are removed when the load operation completes without error. The temporary files are written to a path that can be specified through the

temp-pathname parameter of the LOAD command, or in the *piTempFilesPath* parameter of the db2Load API. The default path is a subdirectory of the database directory.

The temporary files path resides on the server machine and is accessed by the DB2 instance exclusively. Therefore, it is imperative that any path name qualification given to the *temp-pathname* parameter reflects the directory structure of the server, not the client, and that the DB2 instance owner has read and write permission on the path.

Note: In an MPP system, the temporary files path should reside on a local disk, not on an NFS mount. If the path is on an NFS mount, there is significant performance degradation during the load operation.
Attention: The temporary files written to this path must not be tampered with under any circumstances. Doing so causes the load operation to malfunction and places your database in jeopardy.

Load utility log records

The utility manager produces log records associated with a number of DB2 utilities, including the load utility.

The following log records mark the beginning or end of a specific activity during a load operation:

- Load Start. This log record is associated with the beginning of a load operation.
- Load Delete Start. This log record is associated with the beginning of the delete phase in a load operation. The delete phase is started only if there are duplicate primary key values. During the delete phase, each delete operation on a table record, or an index key, is logged.
- Load Delete End. This log record is associated with the end of the delete phase in a load operation. This delete phase is repeated during the rollforward recovery of a successful load operation.
- Load Pending List. This log record is written when a load transaction commits and is used instead of a normal transaction commit log record.

The following list outlines the log records that the load utility creates depending on the size of the input data:

- Two log records are created for every table space extent allocated or deleted by the utility in a DMS table space.
- One log record is created for every chunk of identity values consumed.
- Log records are created for every data row or index key deleted during the delete phase of a load operation.
- Log records are created that maintain the integrity of the index tree when performing a load operation with the ALLOW READ ACCESS and INDEXING MODE INCREMENTAL options specified. The number of records logged is considerably less than a fully logged insertion into the index.

Load overview–partitioned database environments

In a multi-partition database, large amounts of data are located across many database partitions. Distribution keys are used to determine on which database partition each portion of the data resides. The data must be *distributed* before it can be loaded at the correct database partition.

When loading tables in a multi-partition database, the load utility can:

- Distribute input data in parallel
- Load data simultaneously on corresponding database partitions
- Transfer data from one system to another system

Loading data into a multi-partition database takes place in two phases: the *setup phase*, during which database partition resources such as table locks are acquired, and the *load phase*, during which the data is loaded into the database partitions. You can use the ISOLATE_PART_ERRS option of the LOAD command to select how errors are handled during either of these phases, and how errors on one or more of the database partitions affect the load operation on the database partitions that are not experiencing errors.

When loading data into a multi-partition database you can use one of the following modes:

PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. Each file includes a partition header that specifies how the data was distributed across the database partitions, and that the file can be loaded into the database using the LOAD_ONLY mode.

LOAD_ONLY
Data is assumed to be already distributed across the database partitions; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions.

LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed across the database partitions, but the data file does not contain a partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dump file if the `dumpfile` file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition.

ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

Concepts and terminology

The following terminology is used when discussing the behavior and operation of the load utility in a partitioned database environment with multiple database partitions:

- The *coordinator partition* is the database partition to which the user connects in order to perform the load operation. In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, it is assumed that the data file

resides on this database partition unless the CLIENT option of the LOAD command is specified. Specifying CLIENT indicates that the data to be loaded resides on a remotely connected client.

- In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, the *pre-partitioning agent* reads the user data and distributes it in round-robin fashion to the *partitioning agents* which then distribute the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per database partition for any load operation.
- In the PARTITION_AND_LOAD, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, *load agents* run on each output database partition and coordinate the loading of data to that database partition.
- *Load to file agents* run on each output database partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their database partition.
- The SOURCEUSEREXIT option provides a facility through which the load utility can execute a customized script or executable, referred to herein as the *user exit*.

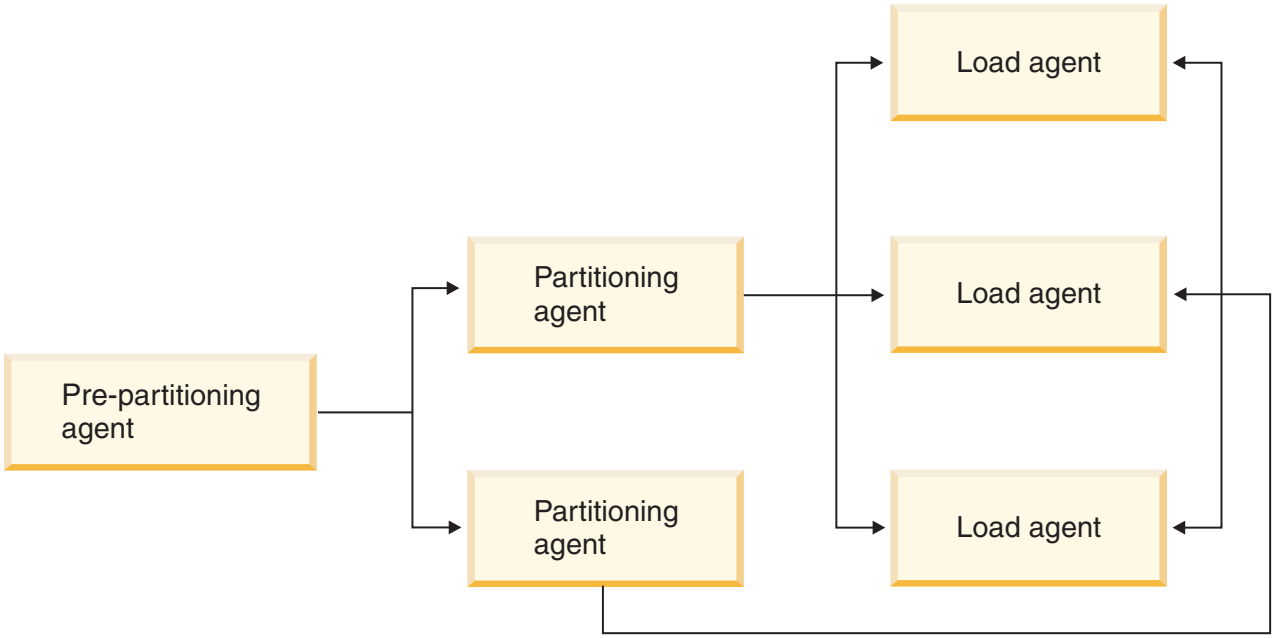


Figure 13. Partitioned Database Load Overview. The source data is read by the pre-partitioning agent, and approximately half of the data is sent to each of two partitioning agents which distribute the data and send it to one of three database partitions. The load agent at each database partition loads the data.

Loading data in a partitioned database environment

Using the load utility to load data into a partitioned database environment.

Before loading a table in a multi-partition database:

1. Ensure that the *svcename* database manager configuration parameter and the **DB2COMM** profile registry variable are set correctly. This is important because the load utility uses TCP/IP to transfer data from the pre-partitioning agent to the partitioning agents, and from the partitioning agents to the loading database partitions.
2. Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be loaded. Since the load utility will issue a COMMIT statement, you should complete all

transactions and release any locks by issuing either a COMMIT or a ROLLBACK statement before beginning the load operation. If the PARTITION_AND_LOAD, PARTITION_ONLY, or ANALYZE mode is being used, the data file that is being loaded must reside on this database partition unless:

a. The CLIENT option has been specified, in which case the data must reside on the client machine;
b. The input source type is CURSOR, in which case there is no input file.

3. Run the Design Advisor to determine the best database partition for each table. For more information, see "The Design Advisor" in *Tuning Database Performance*.

The following restrictions apply when using the load utility to load data in a multi-partition database:

- The location of the input files to the load operation cannot be a tape device.
- The ROWCOUNT option is not supported unless the ANALYZE mode is being used.
- If the target table has an identity column that is needed for distributing and the `identityoverride` file type modifier is not specified, or if you are using multiple database partitions to distribute and then load the data, the use of a SAVECOUNT greater than 0 on the LOAD command is not supported.
- If an identity column forms part of the distribution key, only the PARTITION_AND_LOAD mode is supported.
- The LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes cannot be used with the CLIENT option of the LOAD command.
- The LOAD_ONLY_VERIFY_PART mode cannot be used with the CURSOR input source type.
- The distribution error isolation modes LOAD_ERRS_ONLY and SETUP_AND_LOAD_ERRS cannot be used with the ALLOW READ ACCESS and COPY YES options of the LOAD command.
- Multiple load operations can load data into the same table concurrently if the database partitions specified by the OUTPUT_DBPARTNUMS and PARTITIONING_DBPARTNUMS options do not overlap. For example, if a table is defined on database partitions 0 through 3, one load operation can load data into database partitions 0 and 1 while a second load operation can load data into database partitions 2 and 3.
- Only Non-delimited ASCII (ASC) and Delimited ASCII (DEL) files can be distributed across tables spanning multiple database partitions. PC/IXF files cannot be distributed, however, you can load a PC/IXF file into a table that is distributed over multiple database partitions using the load operation in the LOAD_ONLY_VERIFY_PART mode.

The following examples illustrate how to use the LOAD command to initiate various types of load operations. The database used in the following examples has five database partitions: 0, 1, 2, 3 and 4. Each database partition has a local directory `/db2/data/`. Two tables, TABLE1 and TABLE2, are defined on database partitions 0, 1, 3 and 4. When loading from a client, the user has access to a remote client that is not one of the database partitions.

Loading from a server partition

Distribute and load example

In this scenario you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To load the data from `load.del` into all of the database partitions where TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
```

Note: In this example, default values are used for all of the configuration parameters for partitioned database environments: The MODE parameter default to PARTITION_AND_LOAD, the OUTPUT_DBPARTNUMS options default to all database partitions on which TABLE1 is defined, and the PARTITIONING_DBPARTNUMS defaults to the set of database partitions selected according to the LOAD command rules for choosing database partitions when none are specified.

To perform a load operation where data is distributed over database partitions 3 and 4, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                    PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
```

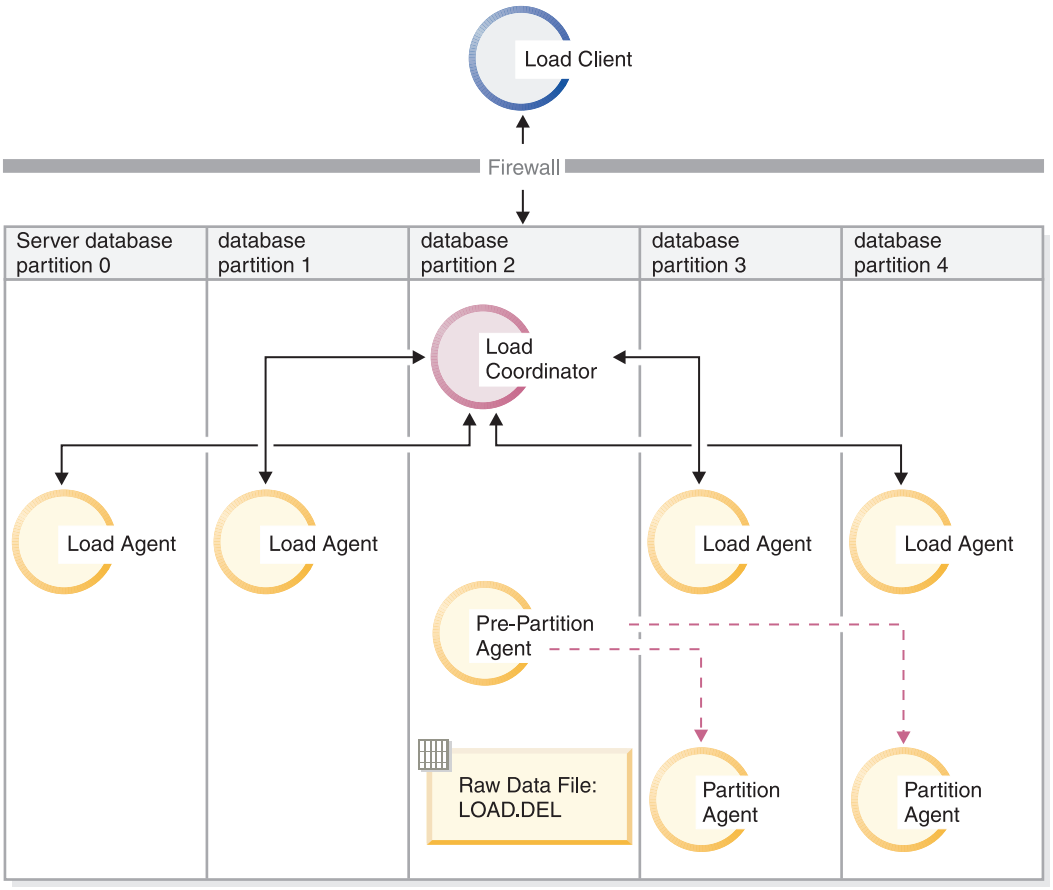


Figure 14. . This diagram illustrates the behavior resulting when the previous command is issued. Data is loaded into database partitions 3 and 4.

Distribute only example

In this scenario you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To distribute (but not load) `load.del` to all the database partitions on which TABLE1 is defined, using database partitions 3 and 4 issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                        PARTITIONED DB CONFIG MODE PARTITION_ONLY
                                              PART_FILE_LOCATION /db2/data
                                              PARTITIONING_DBPARTNUMS (3,4)
```

This results in a file `load.del.xxx` being stored in the `/db2/data` directory on each database partition, where `xxx` is a three-digit representation of the database partition number.

To distribute the `load.del` file to database partitions 1 and 3, using only 1 partitioning agent running on database partition 0 (which is the default for PARTITIONING_DBPARTNUMS), issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                        PARTITIONED DB CONFIG MODE PARTITION_ONLY
                                              PART_FILE_LOCATION /db2/data
                                              OUTPUT_DBPARTNUMS (1,3)
```

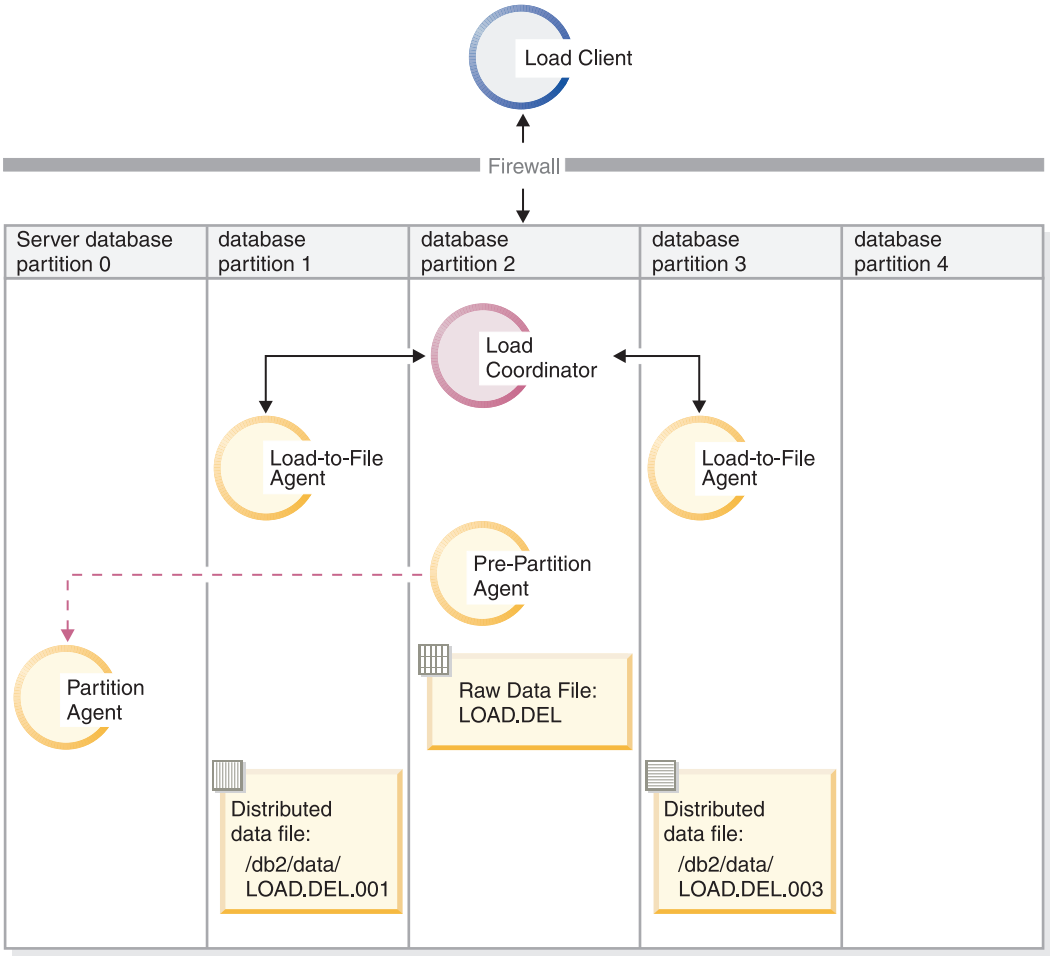


Figure 15. . This diagram illustrates the behavior that results when the previous command is issued. Data is loaded into database partitions 1 and 3, using 1 partitioning agent running on database partition 0.

Load only example

If you have already performed a load operation in the PARTITION_ONLY mode and want to load the partitioned files in the /db2/data directory of each loading database partition to all the database partitions on which TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                      PARTITIONED DB CONFIG MODE LOAD_ONLY
                                            PART_FILE_LOCATION /db2/data
```

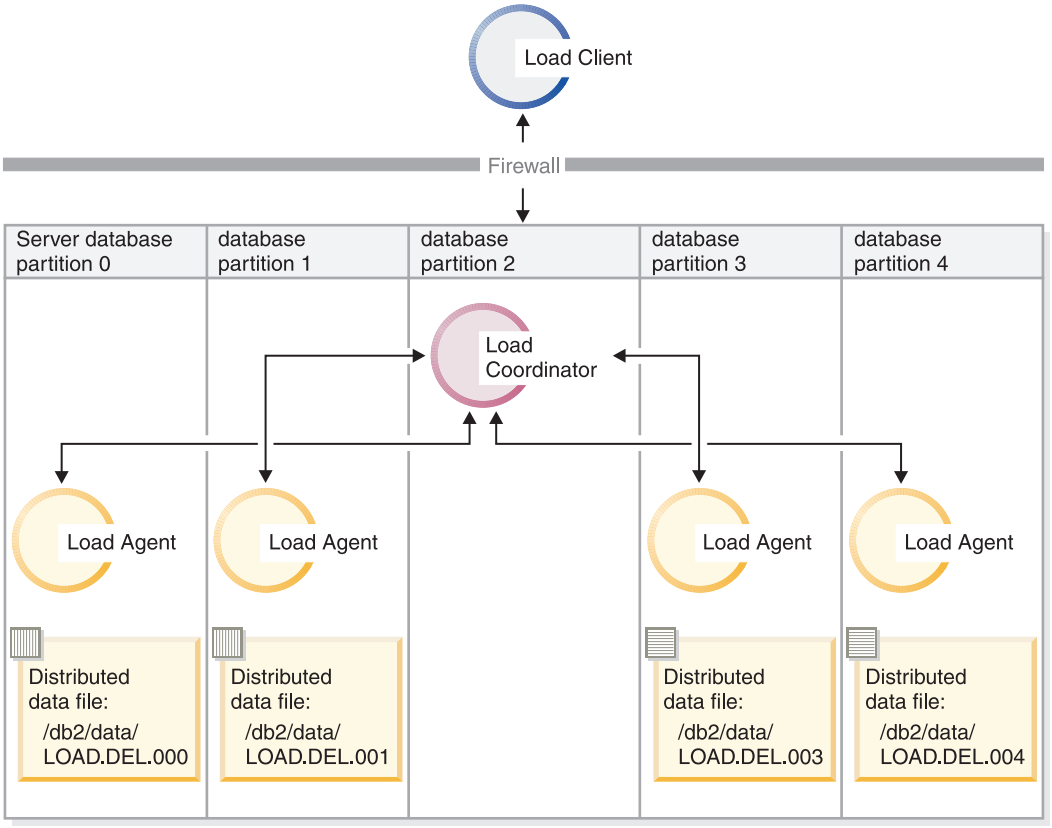


Figure 16. . This diagram illustrates the behavior resulting when the previous command is issued. Distributed data is loaded to all database partitions where TABLE1 is defined.

To load into database partition 4 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                      PARTITIONED DB CONFIG MODE LOAD_ONLY
                                            PART_FILE_LOCATION /db2/data
                                            OUTPUT_DBPARTNUMS (4)
```

Loading pre-distributed files without distribution map headers

The LOAD command can be used to load data files without distribution headers directly into several database partitions. If the data files exist in the /db2/data directory on each database partition where TABLE1 is defined and have the name load.del.xxx, where xxx is the database partition number, the files can be loaded by issuing the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
                       REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
                                             PART_FILE_LOCATION /db2/data
```

To load the data into database partition 1 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
                       REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
                                             PART_FILE_LOCATION /db2/data
                                             OUTPUT_DBPARTNUMS (1)
```

Note: Rows that do not belong on the database partition from which they were loaded are rejected and put into the dumpfile, if one has been specified.

Loading from a remote client to a multi-partition database

To load data into a multi-partition database from a file that is on a remote client, you must specify the CLIENT option of the LOAD command to indicate that the data file is not on a server partition. For example:

```
LOAD CLIENT FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

Note: You cannot use the LOAD_ONLY or LOAD_ONLY_VERIFY_PART modes with the CLIENT option.

Loading from a cursor

As in a single-partition database, you can load from a cursor into a multi-partition database. In this example, for the PARTITION_ONLY and LOAD_ONLY modes, the PART_FILE_LOCATION option must specify a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created or loaded on each output database partition. Multiple files can be created with the specified base name if there are LOB columns in the target table.

To distribute all the rows in the answer set of the statement `SELECT * FROM TABLE1` to a file on each database partition named `/db2/data/select.out.xxx` (where xxx is the database partition number), for future loading into TABLE2, issue the following commands:

```
DECLARE C1 CURSOR FOR SELECT * FROM TABLE1

LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
                       PARTITIONED DB CONFIG MODE PARTITION_ONLY
                       PART_FILE_LOCATION /db2/data/select.out
```

The data files produced by the above operation can then be loaded by issuing the following LOAD command:

```
LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
                       PARTITIONED CB CONFIG MODE LOAD_ONLY
                       PART_FILE_LOCATION /db2/data/select.out
```

Loading data in a partitioned database environment–hints and tips

The following is some information to consider before loading a table in a multi-partition database:

- Familiarize yourself with the load configuration options by using the utility with small amounts of data.

- If the input data is already sorted, or in some chosen order, and you want to maintain that order during the loading process, only one database partition should be used for distributing. Parallel distribution cannot guarantee that the data is loaded in the same order it was received. The load utility chooses a single partitioning agent by default if the `anyorder` modifier is not specified on the LOAD command.
- If large objects (LOBs) are being loaded from separate files (that is, if you are using the `lobsinfile` modifier through the load utility), all directories containing the LOB files must be read-accessible to all the database partitions where loading is taking place. The LOAD *lob-path* parameter must be fully qualified when working with LOBs.
- You can force a job running in a multi-partition database to continue even if the load operation detects (at startup time) that some loading database partitions or associated table spaces or tables are offline, by setting the ISOLATE_PART_ERRS option to SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS.
- Use the STATUS_INTERVAL load configuration option to monitor the progress of a job running in a multi-partition database. The load operation produces messages at specified intervals indicating how many megabytes of data have been read by the pre-partitioning agent. These messages are dumped to the pre-partitioning agent message file. To view the contents of this file during the load operation, connect to the coordinator partition and issue a LOAD QUERY command against the target table.
- Better performance can be expected if the database partitions participating in the distribution process (as defined by the PARTITIONING_DBPARTNUMS option) are different from the loading database partitions (as defined by the OUTPUT_DBPARTNUMS option), since there is less contention for CPU cycles. When loading data into a multi-partition database, invoke the load utility on a database partition that is not participating in either the distributing or the loading operation.
- Specifying the MESSAGES parameter in the LOAD command saves the messages files from the pre-partitioning, partitioning, and load agents for reference at the end of the load operation. To view the contents of these files during a load operation, connect to the desired database partition and issue a LOAD QUERY command against the target table.
- The load utility chooses only one output database partition on which to collect statistics. The RUN_STAT_DBPARTNUM database configuration option can be used to specify the database partition.
- Before loading data in a multi-partition database, run the Design Advisor to determine the best partition for each table. For more information, see "The Design Advisor" in *Tuning Database Performance*.

Troubleshooting

If the load utility is hanging, you can:

- Use the STATUS_INTERVAL parameter to monitor the progress of a multi-partition database load operation. The status interval information is dumped to the pre-partitioning agent message file on the coordinator partition.
- Check the partitioning agent messages file to see the status of the partitioning agent processes on each database partition. If the load is proceeding with no errors, and the TRACE option has been set, there should be trace messages for a number of records in these message files.
- Check the load messages file to see if there are any load error messages.

Note: You must specify the MESSAGES option of the LOAD command in order for these files to exist.

- Interrupt the current load operation if you find errors suggesting that one of the load processes encountered errors.

Monitoring a load operation in a partitioned database environment using the LOAD QUERY command

During a load operation in a partitioned database environment, message files are created by some of the load processes on the database partitions where they are being executed.

The message files store all information, warning, and error messages produced during the execution of the load operation. The load processes that produce message files that can be viewed by the user are the load agent, pre-partitioning agent, and partitioning agent. The content of the message file is only available after the load operation is finished.

You can connect to individual database partitions during a load operation and issue the LOAD QUERY command against the target table. When issued from the CLP, this command displays the contents of all the message files that currently reside on that database partition for the table that is specified in the LOAD QUERY command.

For example, table TABLE1 is defined on database partitions 0 through 3 in database WSDB. You are connected to database partition 0 and issue the following LOAD command:

```
load from load.del of del replace into table1 partitioned db config
    partitioning_dbpartnums (1)
```

This command initiates a load operation that includes load agents running on database partitions 0, 1, 2, and 3; a partitioning agent running on database partition 1; and a pre-partitioning agent running on database partition 0.

Database partition 0 contains one message file for the pre-partitioning agent and one for the load agent on that database partition. To view the contents of these files at the same time, start a new session and issue the following commands from the CLP:

```
set client connect_node 0
connect to wsdb
load query table table1
```

Database partition 1 contains one file for the load agent and one for the partitioning agent. To view the contents of these files, start a new session and issue the following commands from the CLP:

```
set client connect_node 1
connect to wsdb
load query table table1
```

Note: The messages generated by the STATUS_INTERVAL load configuration option appear in the pre-partitioning agent message file. To view these message during a load operation, you must connect to the coordinator partition and issue the LOAD QUERY command.

Saving the contents of message files

If a load operation is initiated through the db2Load API, the messages option (piLocalMsgFileName) must be specified and the message files are brought from the server to the client and stored for you to view.

For multi-partition database load operations initiated from the CLP, the message files are not displayed to the console or retained. To save or view the contents of these files after a multi-partition database load is complete, the MESSAGES option of the LOAD command must be specified. If this option is used, once the load operation is complete the message files on each database partition are transferred to the client machine and stored in files using the base name indicated by the MESSAGES option. For multi-partition database load operations, the name of the file corresponding to the load process that produced it is listed below:

Process Type	File Name
Load Agent	<message-file-name>.LOAD.<dbpartition-number>
Partitioning Agent	<message-file-name>.PART.<dbpartition-number>
Pre-partitioning Agent	<message-file-name>.PREP.<dbpartition-number>

For example, if the MESSAGES option specifies `/wsdb/messages/load`, the load agent message file for database partition 2 is `/wsdb/messages/load.LOAD.002`.

Note: It is strongly recommended that the MESSAGES option be used for multi-partition database load operations initiated from the CLP.

Resuming, restarting, or terminating load operations in a partitioned database environment

The steps you need to take following failed load operations in a partitioned database environment depend on when the failure occurred.

The load process in a multi-partition database consists of two stages:

1. The *setup stage*, during which database partition-level resources such as table locks on output database partitions are acquired

   In general, if a failure occurs during the setup stage, restart and terminate operations are not necessary. What you need to do depends on the error isolation mode that was specified for the failed load operation.

   If the load operation specified that setup stage errors were not to be isolated, the entire load operation is cancelled and the state of the table on each database partition is rolled back to the state it was in prior to the load operation.

   If the load operation specified that setup stage errors were to be isolated, the load operation continues on the database partitions where the setup stage was successful, but the table on each of the failing database partitions is rolled back to the state it was in prior to the load operation. This means that a single load operation can fail at different stages if some partitions fail during the setup stage and others fail during the load stage
2. The *load stage*, during which data is formatted and loaded into tables on the database partitions

   If a load operation fails on at least one database partition during the load stage of a multi-partition database load operation, a load RESTART or load

TERMINATE command must be issued. This is necessary because loading data in a multi-partition database is done through a single transaction.

You should choose a load RESTART if you can fix the problems that caused the failed load to occur. This saves time because if a load restart operation is initiated, the load operation continues from where it left off on all database partitions.

You should choose a load TERMINATE if you want the table returned to the state it was in prior to the initial load operation.

Procedure:

Determining when a load failed

The first thing you need to do if your load operation in a partitioned environment fails is to determine on which partitions it failed and at what stage each of them failed. This is done by looking at the partition summary. If the load command was issued from the CLP, the partition summary is displayed at the end of the load (see example below). If the load command was issued from the db2Load API, the partition summary is contained in the poAgentInfoList field of the db2PartLoadOut structure.

If there is an entry of "LOAD" for "Agent Type", for a given partition, then that partition reached the load stage, otherwise a failure occurred during the setup stage. A negative SQL Code indicates that it failed. In the following example, the load failed on partition 1 during the load stage.

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
.
.
.
```

Resuming, restarting, or terminating a failed load

Only loads with the ISOLATE_PART_ERRS option specifying SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS should fail during the setup stage. For loads that fail on at least one output database partition fail during this stage, you can issue a LOAD REPLACE or LOAD INSERT command. Use the OUTPUT_DBPARTNUMS option to specify only those database partitions on which it failed.

For loads that fail on at least one output database partition during the load stage, issue a load RESTART or load TERMINATE command.

For loads that fail on at least one output database partition during the setup stage and at least one output database partition during the load stage, you need to perform two load operations to resume the failed load–one for the setup stage failures and one for the load stage failures, as previously described. To effectively undo this type of failed load operation, issue a load TERMINATE command. However, after issuing the command, you have to account for all partitions

because no changes were made to the table on the partitions that failed during the setup stage, and all the changes have been undone for the partitions that failed during the load stage.

For example, TABLE1 is defined on database partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load.del of del insert into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

There is a failure on output database partition 1 during the setup stage. Since setup stage errors are isolated, the load operation continues, but there is a failure on partition 3 during the load stage. To resume the load operation, you would issue the following commands:

```
load from load.del of del replace into table1 partitioned db config
output_dbpartnums (1)

load from load.del of del restart into table1 partitioned db config
isolate_part_errs setup_and_load_errs
```

Note: For load restart operations, the options specified in the LOAD RESTART command will be honored, so it is important that they are identical to the ones specified in the original LOAD command.

Migration and version compatibility

The **DB2_PARTITIONEDLOAD_DEFAULT** registry variable can be used to revert to pre-DB2 Universal Database Version 8 load behavior in a multi-partition database

Note: As of Version 9.5, the DB2_PARTITIONEDLOAD_DEFAULT registry variable is deprecated and may be removed in a later release.

Reverting to the pre-DB2 UDB Version 8 behavior of the LOAD command in a multi-partition database, allows you to load a file with a valid distribution header into a single database partition without specifying any extra partitioned database configuration options. You can do this by setting the value of **DB2_PARTITIONEDLOAD_DEFAULT** to NO. You may choose to use this option if you want to avoid modifying existing scripts that issue the LOAD command against single database partitions. For example, to load a distribution file into database partition 3 of a table that resides in a database partition group with four database partitions, issue the following command:

```
db2set DB2_PARTITIONEDLOAD_DEFAULT=NO
```

Then issue the following commands from the DB2 Command Line Processor:

```
CONNECT RESET

SET CLIENT CONNECT_NODE 3

CONNECT TO DB MYDB

LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

In a multi-partition database, when no multi-partition database load configuration options are specified, the load operation takes place on all the database partitions on which the table is defined. The input file does not require a distribution header, and the MODE option defaults to PARTITION_AND_LOAD. To load a single database partition, the OUTPUT_DBPARTNUMS option must be specified.

Reference - Load in a partitioned environment

Load sessions in a partitioned database environment - CLP examples

The following examples demonstrate loading data in a multi-partition database.

The database has four database partitions numbered 0 through 3. Database WSDB is defined on all of the database partitions, and table TABLE1 resides in the default database partition group which is also defined on all of the database partitions.

Example 1

To load data into TABLE1 from the user data file `load.del` which resides on database partition 0, connect to database partition 0 and then issue the following command:

```
load from load.del of del replace into table1
```

If the load operation is successful, the output will be as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that there was one load agent on each database partition and each ran successfully. It also shows that there was one pre-partitioning agent running on the coordinator partition and one partitioning agent running on database partition 1. These processes completed successfully with a normal SQL return code of 0. The statistical summary shows that the pre-partitioning agent read 100,000 rows, the partitioning agent distributed 100,000 rows, and the sum of all rows loaded by the load agents is 100,000.

Example 2

In the following example, data is loaded into TABLE1 in the PARTITION_ONLY mode. The distributed output files is stored on each of the output database partitions in the directory /db/data:

```
load from load.del of del replace into table1 partitioned db config mode
   partition_only part_file_location /db/data
```

The output from the load command is as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD_TO_FILE   000      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   001      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   002      +00000000    Success.
_____________________________________________________
LOAD_TO_FILE   003      +00000000    Success.
_____________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000
```

The output indicates that there was a load-to-file agent running on each output database partition, and these agents ran successfully. There was a pre-partitioning agent on the coordinator partition, and a partitioning agent running on database partition 1. The statistical summary indicates that 100,000 rows were successfully read by the pre-partitioning agent and 100,000 rows were successfully distributed by the partitioning agent. Since no rows were loaded into the table, no summary of the number of rows loaded appears.

Example 3

To load the files that were generated during the PARTITION_ONLY load operation above, issue the following command:

```
load from load.del of del replace into table1 partitioned db config mode
    load_only part_file_location /db/data
```

The output from the load command will be as follows:

```
Agent Type     Node     SQL Code     Result
_____________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________
LOAD           001      +00000000    Success.
_____________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
_____________________________________________________

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
```

```
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that the load agents on each output database partition ran successfully and that the sum of the number of rows loaded by all load agents is 100,000. No summary of rows distributed is indicated since distribution was not performed.

Example 4 - Failed Load Operation

If the following LOAD command is issued:

```
load from load.del of del replace into table1
```

and one of the loading database partitions runs out of space in the table space during the load operation, the following output might be returned:

```
SQL0289N  Unable to allocate new pages in table space "DMS4KT".
SQLSTATE=57011

Agent Type     Node     SQL Code     Result
_____________________________________________________________________
LOAD           000      +00000000    Success.
_____________________________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
_____________________________________________________________________
LOAD           002      +00000000    Success.
_____________________________________________________________________
LOAD           003      +00000000    Success.
_____________________________________________________________________
PARTITION      001      +00000000    Success.
_____________________________________________________________________
PRE_PARTITION  000      +00000000    Success.
_____________________________________________________________________
RESULTS:       3 of 4 LOADs completed successfully.
_____________________________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 0
Rows Rejected               = 0
Rows Partitioned            = 0

Summary of LOAD Agents:
Number of rows read         = 0
Number of rows skipped      = 0
Number of rows loaded       = 0
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 0
```

The output indicates that the load operation returned error SQL0289. The database partition summary indicates that database partition 1 ran out of space. If additional space is added to the containers of the table space on database partition 1, the load operation can be restarted as follows:

```
load from load.del of del restart into table1
```

Load configuration options for partitioned database environments

MODE X

Specifies the mode in which the load operation occurs when loading a multi-partition database. PARTITION_AND_LOAD is the default. Valid values are:

- PARTITION_AND_LOAD. Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- PARTITION_ONLY. Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. For file types other than CURSOR, the format of the output file name on each database partition is `filename.xxx`, where `filename` is the input file name specified in the LOAD command and `xxx` is the 3-digit database partition number. For the CURSOR file type, the name of the output file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Note:

  1. This mode cannot be used for a CLI load operation.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the `identityoverride` file type modifier is specified.
  3. Distribution files generated for file type CURSOR are not compatible between DB2 releases. This means that distribution files of file type CURSOR that were generated in a previous release cannot be loaded using the LOAD_ONLY mode. Similarly, distribution files of file type CURSOR that were generated in the current release cannot be loaded in a future release using the LOAD_ONLY mode.
- LOAD_ONLY. Data is assumed to be already distributed; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. For file types other than CURSOR, the format of the input file name for each database partition should be `filename.xxx`, where `filename` is the name of the file specified in the LOAD command and `xxx` is the 3-digit database partition number. For the CURSOR file type, the name of the input file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Note:

  1. This mode cannot be used for a CLI load operation, or when the CLIENT option of LOAD command is specified.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the `identityoverride` file type modifier is specified.
- LOAD_ONLY_VERIFY_PART. Data is assumed to be already distributed, but the data file does not contain a partition header. The distributing process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dumpfile if the

dumpfile file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning is written to the load message file for that database partition. The format of the input file name for each database partition should be filename.xxx, where filename is the name of the file specified in the LOAD command and xxx is the 3-digit database partition number. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

Note:

1. This mode cannot be used for a CLI load operation, or when the CLIENT option of LOAD command is specified.
2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the identityoverride file type modifier is specified.

- ANALYZE. An optimal distribution map with even distribution across all database partitions is generated.

PART_FILE_LOCATION X

In the PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the distributed files. This location must exist on each database partition specified by the OUTPUT_DBPARTNUMS option. If the location specified is a relative path name, the path is appended to the current directory to create the location for the distributed files.

For the CURSOR file type, this option must be specified, and the location must refer to a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created on each output database partition in the PARTITION_ONLY mode, or the location of the files to be read from for each database partition in the LOAD_ONLY mode. When using the PARTITION_ONLY mode, multiple files can be created with the specified base name if the target table contains LOB columns.

For file types other than CURSOR, if this option is not specified, the current directory is used for the distributed files.

OUTPUT_DBPARTNUMS X

X represents a list of database partition numbers. The database partition numbers represent the database partitions on which the load operation is to be performed. The database partition numbers must be a subset of the database partitions on which the table is defined. All database partitions are selected by default. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)).

PARTITIONING_DBPARTNUMS X

X represents a list of database partition numbers that are used in the distribution process. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)). The database partitions specified for the distribution process can be different from the database partitions being loaded. If PARTITIONING_DBPARTNUMS is not specified, the load utility determines how many database partitions are needed and which database partitions to use in order to achieve optimal performance.

If the anyorder file type modifier is not specified in the LOAD command, only one partitioning agent is used in the load session. Furthermore, if there is only one database partition specified for the OUTPUT_DBPARTNUMS option, or the coordinator partition of the load operation is not an element of OUTPUT_DBPARTNUMS, the coordinator partition of the load operation is used in the distribution process. Otherwise, the first database partition (not the coordinator partition) in OUTPUT_DBPARTNUMS is used in the distribution process.

If the anyorder file type modifier is specified, the number of database partitions used in the distribution process is determined as follows: (number of partitions in OUTPUT_DBPARTNUMS/4 + 1).

MAX_NUM_PART_AGENTS X

Specifies the maximum numbers of partitioning agents to be used in a load session. The default is 25.

ISOLATE_PART_ERRS X

Indicates how the load operation reacts to errors that occur on individual database partitions. The default is LOAD_ERRS_ONLY, unless both the ALLOW READ ACCESS and COPY YES options of the LOAD command are specified, in which case the default is NO_ISOLATION. Valid values are:

- SETUP_ERRS_ONLY. Errors that occur on a database partition during setup, such as problems accessing a database partition, or problems accessing a table space or table on a database partition, cause the load operation to stop on the failing database partitions but to continue on the remaining database partitions. Errors that occur on a database partition while data is being loaded cause the entire operation to fail.
- LOAD_ERRS_ONLY. Errors that occur on a database partition during setup cause the entire load operation to fail. If an error occurs while data is being loaded, the load operation will stop on the database partition where the error occurred. The load operation continues on the remaining database partitions until a failure occurs or until all the data is loaded. The newly loaded data will not be visible until a load restart operation is performed and completes successfully.

  Note: This mode cannot be used when both the ALLOW READ ACCESS and the COPY YES options of the LOAD command are specified.
- SETUP_AND_LOAD_ERRS. In this mode, database partition-level errors during setup or loading data cause processing to stop only on the affected database partitions. As with the LOAD_ERRS_ONLY mode, when partition errors do occur while data is loaded, newly loaded data will not be visible until a load restart operation is performed and completes successfully.

  Note: This mode cannot be used when both the ALLOW READ ACCESS and the COPY YES options of the LOAD command are specified.
- NO_ISOLATION. Any error during the load operation causes the load operation to fail.

STATUS_INTERVAL X

X represents how often you are notified of the volume of data that has been read. The unit of measurement is megabytes (MB). The default is 100 MB. Valid values are whole numbers from 1 to 4000.

PORT_RANGE X

X represents the range of TCP ports used to create sockets for internal communications. The default range is from 6000 to 6063. If defined at the time of invocation, the value of the **DB2ATLD_PORTS** registry variable replaces the value of the PORT_RANGE load configuration option. For the **DB2ATLD_PORTS** registry variable, the range should be provided in the following format:

```
<lower-port-number:higher-port-number>
```

From the CLP, the format is:

```
( lower-port-number, higher-port-number )
```

CHECK_TRUNCATION

Specifies that the program should check for truncation of data records at input/output. The default behavior is that data is not checked for truncation at input/output.

MAP_FILE_INPUT X

X specifies the input file name for the distribution map. This parameter must be specified if the distribution map is customized, as it points to the file containing the customized distribution map. A customized distribution map can be created by using the db2gpmap program to extract the map from the database system catalog table, or by using the ANALYZE mode of the LOAD command to generate an optimal map. The map generated by using the ANALYZE mode must be moved to each database partition in your database before the load operation can proceed.

MAP_FILE_OUTPUT X

X represents the output filename for the distribution map. The output file is created on the database partition issuing the LOAD command assuming that database partition is participating in the database partition group where partitioning is performed. If the LOAD command is invoked on a database partition that is not participating in partitioning (as defined by PARTITIONING_DBPARTNUMS), the output file is created at the first database partition defined with the PARTITIONING_DBPARTNUMS parameter. Consider the following partitioned database environment setup:

```
1 serv1 0
2 serv1 1
3 serv2 0
4 serv2 1
5 serv3 0
```

Running the following LOAD command on serv3, creates the distribution map on serv1.

```
LOAD FROM file OF ASC METHOD L ( ...) INSERT INTO table CONFIG
MODE ANALYZE PARTITIONING_DBPARTNUMS(1,2,3,4)
MAP_FILE_OUTPUT '/home/db2user/distribution.map'
```

This parameter should be used when the ANALYZE mode is specified. An optimal distribution map with even distribution across all database partitions is generated. If this parameter is not specified and the ANALYZE mode is specified, the program exits with an error.

TRACE X

Specifies the number of records to trace when you require a review of a dump of the data conversion process and the output of the hashing values. The default is 0.

NEWLINE
Used when the input data file is an ASC file with each record delimited by a new line character and the `reclen` file type modifier is specified in the LOAD command. When this option is specified, each record is checked for a new line character. The record length, as specified in the `reclen` file type modifier, is also checked.

DISTFILE X
If this option is specified, the load utility generates a database partition distribution file with the given name. The database partition distribution file contains 4096 integers: one for each entry in the target table's distribution map. Each integer in the file represents the number of rows in the input files being loaded that hashed to the corresponding distribution map entry. This information can help you identify skew in your data and also help you decide whether a new distribution map should be generated for the table using the ANALYZE mode of the utility. If this option is not specified, the default behavior of the load utility is to not generate the distribution file.

Note: When this option is specified, a maximum of one distribution agent is used for the load operation. Even if you explicitly request multiple distribution agents, only one is used.

OMIT_HEADER
Specifies that a distribution map header should not be included in the distribution file. If not specified, a header is generated.

RUN_STAT_DBPARTNUM X
If the STATISTICS YES parameter is specified in the LOAD command, statistics are collected only on one database partition. This parameter specifies on which database partition to collect statistics. If the value is -1 or not specified at all, statistics are collected on the first database partition in the output database partition list.

Reference - Load

LOAD

Loads data into a DB2 table. Data residing on the server can be in the form of a file, tape, or named pipe. If the `COMPRESS` attribute for the table is set to `YES`, the data loaded will be subject to compression on every data and database partition for which a dictionary already exists in the table.

Quick link to "File type modifiers for the load utility" on page 217.

Restrictions

The load utility does not support loading data at the hierarchy level. The load utility is not compatible with range-clustered tables.

Scope

This command can be issued against multiple database partitions in a single request.

Authorization

One of the following:

- *sysadm*
- *dbadm*
- LOAD authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the load fails and an error (SQLSTATE 5U014) is returned.
- To load data into a table that has protected rows, the session authorization id must hold a security label that meets these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access or for all access

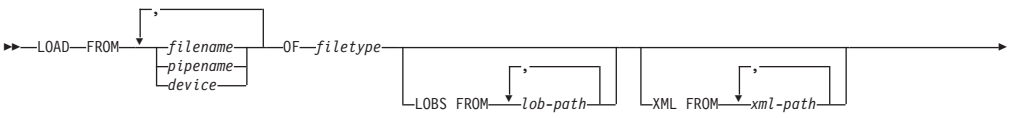
  If the session authorization id does not hold such a security label then the load fails and an error (SQLSTATE 5U014) is returned. This security label is used to protect a loaded row if the session authorization ID's LBAC credentials do not allow it to write to the security label that protects that row in the data. This does not happen, however, when the security policy protecting the table was created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option of the CREATE SECURITY POLICY statement. In this case the load fails and an error (SQLSTATE 42519) is returned.
- If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.

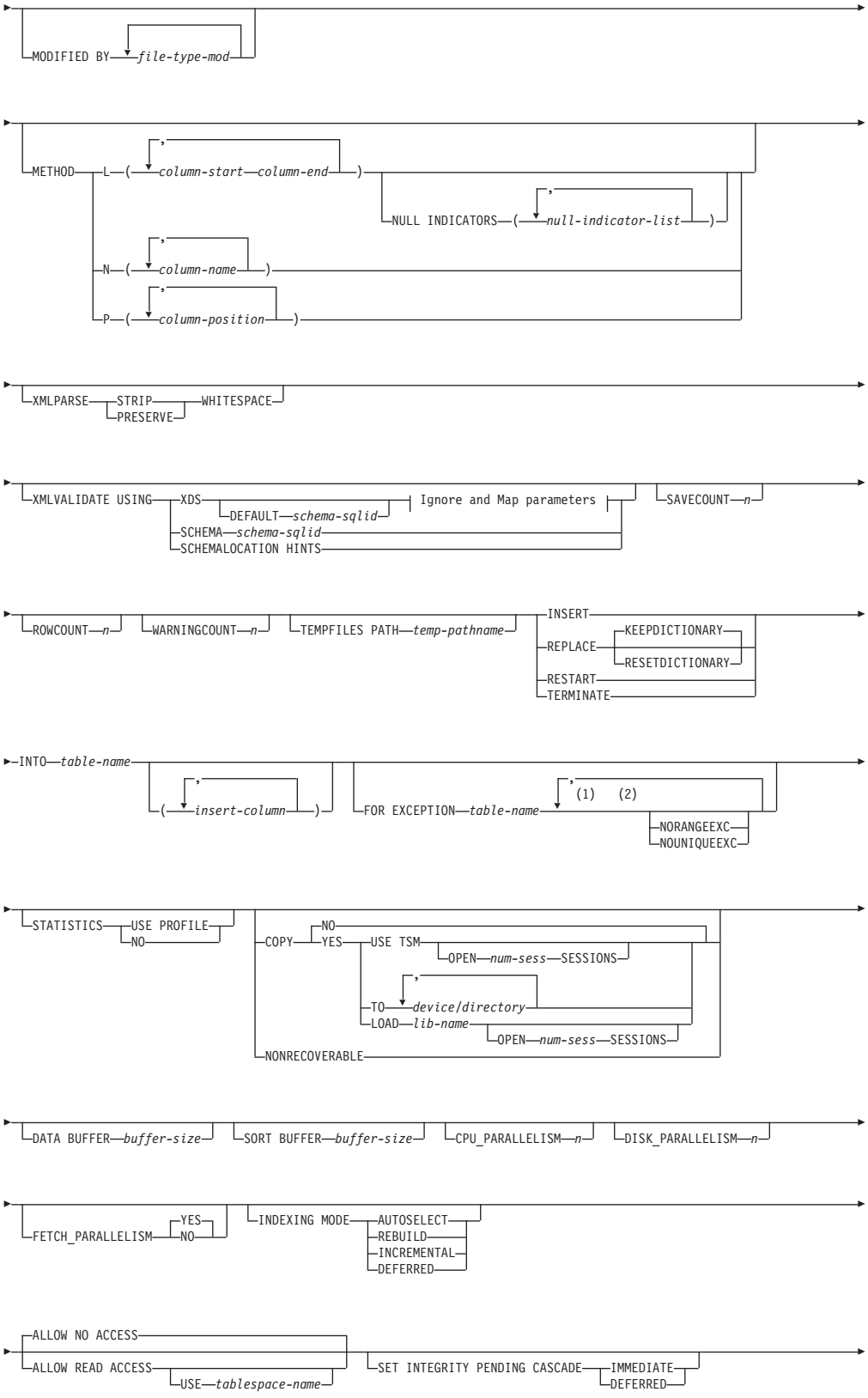
Since all load processes (and all DB2 server processes, in general) are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

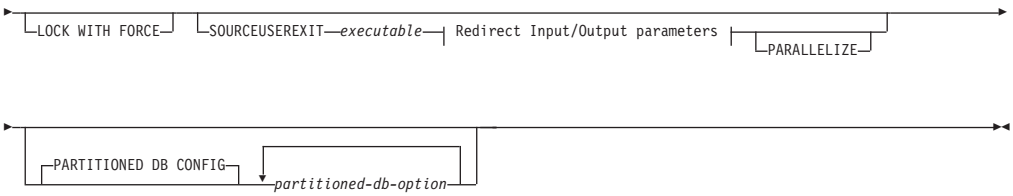
Required connection

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

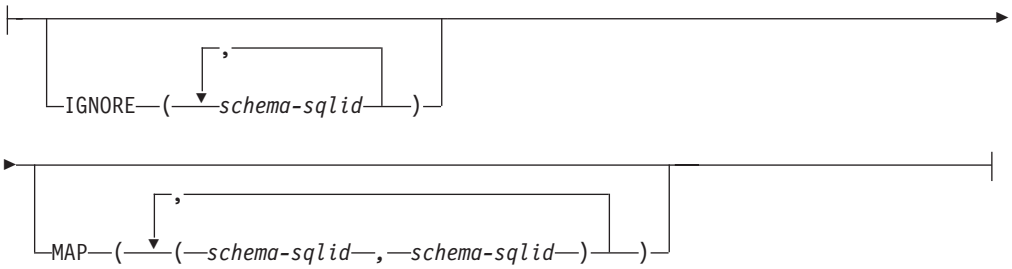
Command syntax



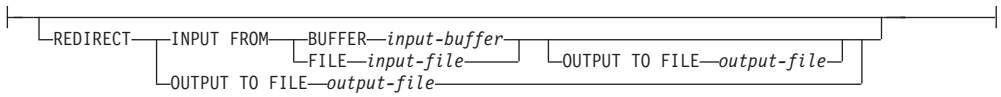

MODIFIED BY file-type-mod
METHOD L (column-start column-end)
NULL INDICATORS (null-indicator-list)
N (column-name)
P (column-position)
XMLPARSE STRIP WHITESPACE
PRESERVE
XMLVALIDATE USING XDS
DEFAULT schema-sqlid
SCHEMA schema-sqlid
SCHEMALOCATION HINTS
Ignore and Map parameters
SAVECOUNT n
ROWCOUNT n
WARNINGCOUNT n
TEMPFILES PATH temp-pathname
INSERT
REPLACE
KEEPDICTIONARY
RESETDICTIONARY
RESTART
TERMINATE
INTO table-name
(insert-column)
FOR EXCEPTION table-name (1) (2)
NORANGEEXC
NOUNIQUEEXC
STATISTICS USE PROFILE
NO
COPY NO
YES USE TSM
OPEN num-sess SESSIONS
TO device/directory
LOAD lib-name
OPEN num-sess SESSIONS
NONRECOVERABLE
DATA BUFFER buffer-size
SORT BUFFER buffer-size
CPU_PARALLELISM n
DISK_PARALLELISM n
FETCH_PARALLELISM YES
NO
INDEXING MODE AUTOSELECT
REBUILD
INCREMENTAL
DEFERRED
ALLOW NO ACCESS
ALLOW READ ACCESS
USE tablespace-name
SET INTEGRITY PENDING CASCADE IMMEDIATE
DEFERRED



Ignore and Map parameters:



Redirect Input/Output parameters:



Notes:

1 These keywords can appear in any order.

2 Each of these keywords can only appear once.

Command parameters

FROM *filename* | *pipename* | *device*

Note:

1. If data is exported into a file using the *EXPORT command using the ADMIN_CMD procedure*, the data file is owned by the fenced user ID. This file is not usually accessible by the instance owner. To run the LOAD from CLP or the ADMIN_CMD procedure, the data file must be accessible by the instance owner ID, so read access to the data file must be granted to the instance owner.
2. Loading data from multiple IXF files is supported if the files are physically separate, but logically one file. It is *not* supported if the files are both logically and physically separate. (Multiple physical files would be considered logically one if they were all created with one invocation of the EXPORT command.)

OF *filetype*

Specifies the format of the data:

- ASC (non-delimited ASCII format).
- DEL (delimited ASCII format).
- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2.

- CURSOR (a cursor declared against a SELECT or VALUES statement).

LOBS FROM *lob-path*
The path to the data files containing LOB values to be loaded. The path must end with a slash (/). The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the LOBSINFILE behavior.

This option is ignored when specified in conjunction with the CURSOR file type.

MODIFIED BY *file-type-mod*
Specifies file type modifier options. See "File type modifiers for the load utility" on page 217.

METHOD

L Specifies the start and end column numbers from which to load data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1. This method can only be used with ASC files, and is the only valid method for that file type.

NULL INDICATORS *null-indicator-list*
This option can only be used when the METHOD L parameter is specified; that is, the input file is an ASC file). The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the METHOD L parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of Y in the NULL indicator column specifies that the column data is NULL. Any character *other than* Y in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the METHOD L option will be loaded.

The NULL indicator character can be changed using the MODIFIED BY option.

N Specifies the names of the columns in the data file to be loaded. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the METHOD N list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method N (F2, F1, F4, F3)` is a valid request, while `method N (F2, F1)` is not valid. This method can only be used with file types IXF or CURSOR.

P Specifies the field numbers (numbered from 1) of the input data fields to be loaded. Each table column that is not nullable should have a corresponding entry in the METHOD P list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method P (2, 1, 4, 3)` is a valid request, while `method P (2, 1)` is not

valid. This method can only be used with file types IXF, DEL, or CURSOR, and is the only valid method for the DEL file type.

XML FROM *xml-path*
Specifies one or more paths that contain the XML files. XDSs are contained in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the XML column.

XMLPARSE
Specifies how XML documents are parsed. If this option is not specified, the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE
Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE
Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the XMLVALIDATE option is invoked with the USING XDS clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the DEFAULT clause.

The DEFAULT, IGNORE, and MAP clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the DEFAULT clause, it will not be ignored if also specified by the IGNORE clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the MAP clause, it will not be re-mapped if also specified in the second part of another MAP clause pair.

USING SCHEMA *schema-sqlid*
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the USING SCHEMALOCATION HINTS clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the XMLVALIDATE option below.

IGNORE *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The IGNORE clause specifies a list of one or more schemas to ignore if

they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for a loaded XML document, and the schema identified by the SCH attribute is included in the list of schemas to IGNORE, then no schema validation will occur for the loaded XML document.

Note:

If a schema is specified in the IGNORE clause, it cannot also be present in the left side of a schema pair in the MAP clause.

The IGNORE clause applies only to the XDS. A schema that is mapped by the MAP clause will not be subsequently ignored if specified by the IGNORE clause.

DEFAULT *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The schema specified through the DEFAULT clause identifies a schema to use for validation when the XML Data Specifier (XDS) of a loaded XML document does not contain an SCH attribute identifying an XML Schema.

The DEFAULT clause takes precedence over the IGNORE and MAP clauses. If an XDS satisfies the DEFAULT clause, the IGNORE and MAP specifications will be ignored.

MAP *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. Use the MAP clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each loaded XML document. The MAP clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the MAP clause, it cannot also be specified in the IGNORE clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

SAVECOUNT *n*
Specifies that the load utility is to establish consistency points after every *n* rows. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using LOAD QUERY. If the value of *n* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is zero, meaning that no consistency points will be established, unless necessary.

This option is ignored when specified in conjunction with the CURSOR file type.

ROWCOUNT *n*

Specifies the number of *n* physical records in the file to be loaded. Allows a user to load only the first *n* rows in a file.

WARNINGCOUNT *n*

Stops the load operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If *n* is zero, or this option is not specified, the load operation will continue regardless of the number of warnings issued. If the load operation is stopped because the threshold of warnings was encountered, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

TEMPFILES PATH *temp-pathname*

Specifies the name of the path to be used when creating temporary files during a load operation, and should be fully qualified according to the server database partition.

Temporary files take up file system space. Sometimes, this space requirement is quite substantial. Following is an estimate of how much file system space should be allocated for all temporary files:

- 136 bytes for each message that the load utility generates
- 15 KB overhead if the data file contains long field data or LOBs. This quantity can grow significantly if the INSERT option is specified, and there is a large amount of long field or LOB data already in the table.

INSERT

One of four modes under which the load utility can execute. Adds the loaded data to the table without changing the existing table data.

REPLACE

One of four modes under which the load utility can execute. Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

KEEPDICTIONARY

An existing compression dictionary is preserved across the LOAD REPLACE operation. Provided the table COMPRESS attribute is YES, the newly replaced data is subject to being compressed using the dictionary that existed prior to the invocation of the load. If no dictionary previously existed in the table, a new dictionary is built using the data that is being replaced into the table as long as the table COMPRESS attribute is YES. The amount of data that is required to build the compression dictionary in this case is subject to the policies of ADC. This data is populated into the table as uncompressed. Once the dictionary is inserted into the table, the remaining data to be loaded is subject to being compressed with this dictionary. This is the default parameter. For summary, see Table 1 below.

Table 27. LOAD REPLACE KEEPDICTIONARY

Compress	Dictionary exists	Result outcome
Y	Y	Preserve dictionary; all input rows are subject to compression with existing dictionary.
Y	N	Insert new dictionary into table only if sufficient user data exists; remaining rows subject to compression after dictionary built.
N	Y	Preserve dictionary; all input rows are not compressed.
N	N	No effect; all rows not compressed.

RESETDICTIONARY

This directive instructs LOAD REPLACE processing to build a new dictionary for the table data object provided that the table COMPRESS attribute is YES. If the COMPRESS attribute is NO and a dictionary was already present in the table it will be removed and no new dictionary will be inserted into the table. A compression dictionary can be built with just one user record. If the loaded data set size is zero and if there is a pre-existing dictionary, the dictionary will not be preserved. The amount of data required to build a dictionary with this directive is not subject to the policies of ADC. For summary, see Table 2 below.

Table 28. LOAD REPLACE RESETDICTIONARY

Compress	Dictionary exists	Result outcome
Y	Y	Build a new dictionary*; remaining rows to be loaded are subject to compression after dictionary built.
Y	N	Build new dictionary; remaining rows subject to compression after dictionary built.
N	Y	Remove dictionary; all input rows are not compressed.
N	N	No effect; all rows not compressed.

* If a dictionary exists and the compression attribute is enabled, but there are no records to load into the table partition, a new dictionary cannot be built and the RESETDICTIONARY operation will not keep the existing dictionary.

TERMINATE

One of four modes under which the load utility can execute. Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects might be marked as invalid, in which case index rebuild will automatically take place at next access). If the load operation being terminated is a LOAD REPLACE, the table will be truncated to an empty table after the LOAD TERMINATE operation. If the load operation being terminated is a LOAD INSERT, the table will retain all of its original records after the LOAD TERMINATE operation. For summary of dictionary management, see Table 3 below.

The LOAD TERMINATE option will not remove a backup pending state from table spaces.

RESTART
One of four modes under which the load utility can execute. Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase. For summary of dictionary management, see Table 4 below.

INTO *table-name*
Specifies the database table into which the data is to be loaded. This table cannot be a system table or a declared temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form `schema.tablename`. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

insert-column
Specifies the table column into which the data is to be inserted.

The load utility cannot parse columns whose names contain one or more spaces. For example,

will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

FOR EXCEPTION *table-name*
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

Information that is written to the exception table is *not* written to the dump file. In a partitioned database environment, an exception table must be defined for those database partitions on which the loading table is defined. The dump file, otherwise, contains rows that cannot be loaded because they are invalid or have syntax errors.

NORANGEEXC
Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

NOUNIQUEEXC
Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

STATISTICS USE PROFILE
Instructs load to collect statistics during the load according to the profile defined for this table. This profile must be created before load is executed. The profile is created by the RUNSTATS command. If the profile does not exist and load is instructed to collect statistics according to the profile, a warning is returned and no statistics are collected.

STATISTICS NO
Specifies that no statistics are to be collected, and that the statistics in the catalogs are not to be altered. This is the default.

COPY NO
Specifies that the table space in which the table resides will be placed in backup pending state if forward recovery is enabled (that is, *logretain* or *userexit* is on). The COPY NO option will also put the table space state into the Load in Progress table space state. This is a transient state that will disappear when the load completes or aborts. The data in any table in the table space cannot be updated or deleted until a table space backup or a

full database backup is made. However, it is possible to access the data in any table by using the SELECT statement.

LOAD with COPY NO on a recoverable database leaves the table spaces in a backup pending state. For example, performing a LOAD with COPY NO and INDEXING MODE DEFERRED will leave indexes needing a refresh. Certain queries on the table might require an index scan and will not succeed until the indexes are refreshed. The index cannot be refreshed if it resides in a table space which is in the backup pending state. In that case, access to the table will not be allowed until a backup is taken. Index refresh is done automatically by the database when the index is accessed by a query. If one of COPY NO, COPY YES, or NONRECOVERABLE is not specified, and the database is recoverable (**logretain** or **logarchmeth1** is enabled), then COPY NO is the default.

COPY YES
Specifies that a copy of the loaded data will be saved. This option is invalid if forward recovery is disabled (both *logretain* and *userexit* are off).

USE TSM
Specifies that the copy will be stored using Tivoli Storage Manager (TSM).

OPEN *num-sess* **SESSIONS**
The number of I/O sessions to be used with TSM or the vendor product. The default value is 1.

TO *device/directory*
Specifies the device or directory on which the copy image will be created.

LOAD *lib-name*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path where the user exit programs reside.

NONRECOVERABLE
Specifies that the load transaction is to be marked as non-recoverable and that it will not be possible to recover it by a subsequent roll forward action. The roll forward utility will skip the transaction and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward operation is completed, such a table can only be dropped or restored from a backup (full or table space) taken after a commit point following the completion of the non-recoverable load operation.

With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. If one of COPY NO, COPY YES, or NONRECOVERABLE is not specified, and the database is not recoverable (**logretain** or **logarchmeth1** is not enabled), then NONRECOVERABLE is the default.

WITHOUT PROMPTING
Specifies that the list of data files contains all the files that are to be loaded, and that the devices or directories listed are sufficient for the entire load operation. If a continuation input file is not found, or the copy targets are filled before the load operation finishes, the load operation will fail, and the table will remain in load pending state.

DATA BUFFER *buffer-size*
Specifies the number of 4 KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the minimum required resource is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the *util_heap_sz* database configuration parameter.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

SORT BUFFER *buffer-size*
This option specifies a value that overrides the SORTHEAP database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the INDEXING MODE parameter is not specified as DEFERRED. The value that is specified cannot exceed the value of SORTHEAP. This parameter is useful for throttling the sort memory that is used when loading tables with many indexes without changing the value of SORTHEAP, which would also affect general query processing.

CPU_PARALLELISM *n*
Specifies the number of processes or threads that the load utility will create for parsing, converting, and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, or has not been specified, the load utility uses an intelligent default value (usually based on the number of CPUs available) at run time.

Note:

1. If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs or the value specified by the user.
2. Specifying a small value for the SAVECOUNT parameter causes the loader to perform many more I/O operations to flush both data and table metadata. When CPU_PARALLELISM is greater than one, the flushing operations are asynchronous, permitting the loader to exploit the CPU. When CPU_PARALLELISM is set to one, the loader waits on I/O during consistency points. A load operation with CPU_PARALLELISM set to two, and SAVECOUNT set to 10 000, completes faster than the same operation with CPU_PARALLELISM set to one, even though there is only one CPU.

DISK_PARALLELISM *n*
Specifies the number of processes or threads that the load utility will create for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

FETCH_PARALLELISM YES | NO
When performing a load from a cursor where the cursor is declared using the DATABASE keyword, or when using the API `sqlu_remotefetch_entry` media entry, and this option is set to YES, the load utility attempts to parallelize fetching from the remote data source if possible. If set to NO,

no parallel fetching is performed. The default value is YES. For more information, see *Moving data using the CURSOR file type*.

INDEXING MODE

Specifies whether the load utility is to rebuild indexes or to extend them incrementally. Valid values are:

AUTOSELECT

The load utility will automatically decide between REBUILD or INCREMENTAL mode. The decision is based on the amount of data being loaded and the depth of the index tree. Information relating to the depth of the index tree is stored in the index object. RUNSTATS is not required to populate this information. AUTOSELECT is the default indexing mode.

REBUILD

All indexes will be rebuilt. The utility must have sufficient resources to sort all index key parts for both old and appended table data.

INCREMENTAL

Indexes will be extended with new data. This approach consumes index free space. It only requires enough sort space to append index keys for the inserted records. This method is only supported in cases where the index object is valid and accessible at the start of a load operation (it is, for example, not valid immediately following a load operation in which the DEFERRED mode was specified). If this mode is specified, but not supported due to the state of the index, a warning is returned, and the load operation continues in REBUILD mode. Similarly, if a load restart operation is begun in the load build phase, INCREMENTAL mode is not supported.

Incremental indexing is not supported when all of the following conditions are true:

- The LOAD COPY option is specified (*logarchmeth1* with the `USEREXIT` or `LOGRETAIN` option).
- The table resides in a DMS table space.
- The index object resides in a table space that is shared by other table objects belonging to the table being loaded.

To bypass this restriction, it is recommended that indexes be placed in a separate table space.

DEFERRED

The load utility will not attempt index creation if this mode is specified. Indexes will be marked as needing a refresh. The first access to such indexes that is unrelated to a load operation might force a rebuild, or indexes might be rebuilt when the database is restarted. This approach requires enough sort space for all key parts for the largest index. The total time subsequently taken for index construction is longer than that required in REBUILD mode. Therefore, when performing multiple load operations with deferred indexing, it is advisable (from a performance viewpoint) to let the last load operation in the sequence perform an index rebuild, rather than allow indexes to be rebuilt at first non-load access.

Deferred indexing is only supported for tables with non-unique indexes, so that duplicate keys inserted during the load phase are not persistent after the load operation.

ALLOW NO ACCESS
Load will lock the target table for exclusive access during the load. The table state will be set to Load In Progress during the load. ALLOW NO ACCESS is the default behavior. It is the only valid option for LOAD REPLACE.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending state.

ALLOW READ ACCESS
Load will lock the target table in a share mode. The table state will be set to both Load In Progress and Read Access. Readers can access the non-delta portion of the data while the table is being load. In other words, data that existed before the start of the load will be accessible by readers to the table, data that is being loaded is not available until the load is complete. LOAD TERMINATE or LOAD RESTART of an ALLOW READ ACCESS load can use this option; LOAD TERMINATE or LOAD RESTART of an ALLOW NO ACCESS load cannot use this option. Furthermore, this option is not valid if the indexes on the target table are marked as requiring a rebuild.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress, and Read Access. At the end of the load, the table state Load In Progress will be removed but the table states Set Integrity Pending and Read Access will remain. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending. While the table is in Set Integrity Pending and Read Access states, the non-delta portion of the data is still accessible to readers, the new (delta) portion of the data will remain inaccessible until the SET INTEGRITY statement has completed. A user can perform multiple loads on the same table without issuing a SET INTEGRITY statement. Only the original (checked) data will remain visible, however, until the SET INTEGRITY statement is issued.

ALLOW READ ACCESS also supports the following modifiers:

USE *tablespace-name*
If the indexes are being rebuilt, a shadow copy of the index is built in table space *tablespace-name* and copied over to the original table space at the end of the load during an INDEX COPY PHASE. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same table space as the index object. If the shadow copy is created in the same table space as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different table space from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load during the INDEX COPY PHASE.

Without this option the shadow index is built in the same table space as the original. Since both the original index and shadow index by default reside in the same table space simultaneously,

there might be insufficient space to hold both indexes within one table space. Using this option ensures that you retain enough table space for the indexes.

This option is ignored if the user does not specify INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT. This option will also be ignored if INDEXING MODE AUTOSELECT is chosen and load chooses to incrementally update the index.

SET INTEGRITY PENDING CASCADE

If LOAD puts the table into Set Integrity Pending state, the SET INTEGRITY PENDING CASCADE option allows the user to specify whether or not Set Integrity Pending state of the loaded table is immediately cascaded to all descendents (including descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables).

IMMEDIATE

Indicates that Set Integrity Pending state is immediately extended to all descendent foreign key tables, descendent immediate materialized query tables and descendent staging tables. For a LOAD INSERT operation, Set Integrity Pending state is not extended to descendent foreign key tables even if the IMMEDIATE option is specified.

When the loaded table is later checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement), descendent foreign key tables that were placed in Set Integrity Pending Read Access state will be put into Set Integrity Pending No Access state.

DEFERRED

Indicates that only the loaded table will be placed in the Set Integrity Pending state. The states of the descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged.

Descendent foreign key tables might later be implicitly placed in Set Integrity Pending state when their parent tables are checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement). Descendent immediate materialized query tables and descendent immediate staging tables will be implicitly placed in Set Integrity Pending state when one of its underlying tables is checked for integrity violations. A warning (SQLSTATE 01586) will be issued to indicate that dependent tables have been placed in Set Integrity Pending state. See the Notes section of the SET INTEGRITY statement in the SQL Reference for when these descendent tables will be put into Set Integrity Pending state.

If the SET INTEGRITY PENDING CASCADE option is not specified:

- Only the loaded table will be placed in Set Integrity Pending state. The state of descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged, and can later be implicitly put into Set Integrity Pending state when the loaded table is checked for constraint violations.

If LOAD does not put the target table into Set Integrity Pending state, the SET INTEGRITY PENDING CASCADE option is ignored.

LOCK WITH FORCE
The utility acquires various locks including table locks in the process of loading. Rather than wait, and possibly timeout, when acquiring a lock, this option allows load to force off other applications that hold conflicting locks on the target table. Applications holding conflicting locks on the system catalog tables will not be forced off by the load utility. Forced applications will roll back and release the locks the load utility needs. The load utility can then proceed. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

ALLOW NO ACCESS loads might force applications holding conflicting locks at the start of the load operation. At the start of the load the utility can force applications that are attempting to either query or modify the table.

ALLOW READ ACCESS loads can force applications holding conflicting locks at the start or end of the load operation. At the start of the load the load utility can force applications that are attempting to modify the table. At the end of the load operation, the load utility can force applications that are attempting to either query or modify the table.

SOURCEUSEREXIT*executable*
Specifies an executable filename which will be called to feed data into the utility.

REDIRECT

INPUT FROM

BUFFER *input-buffer*
The stream of bytes specified in *input-buffer* is passed into the STDIN file descriptor of the process executing the given executable.

FILE *input-file*
The contents of this client-side file are passed into the STDIN file descriptor of the process executing the given executable.

OUTPUT TO

FILE *output-file*
The STDOUT and STDERR file descriptors are captured to the fully qualified server-side file specified.

PARALLELIZE
Increases the throughput of data coming into the load utility by invoking multiple user exit processes simultaneously. This option is only applicable in multi-partition database environments and is ignored in single-partition database environments.

For more information, see *Moving data using a customized application (user exit)*.

PARTITIONED DB CONFIG *partitioned-db-option*
Allows you to execute a load into a table distributed across multiple database partitions. The PARTITIONED DB CONFIG parameter allows you to specify partitioned database-specific configuration options. The *partitioned-db-option* values can be any of the following:

```
PART_FILE_LOCATION x
OUTPUT_DBPARTNUMS x
PARTITIONING_DBPARTNUMS x
MODE x
MAX_NUM_PART_AGENTS x
ISOLATE_PART_ERRS x
STATUS_INTERVAL x
PORT_RANGE x
CHECK_TRUNCATION
MAP_FILE_INPUT x
MAP_FILE_OUTPUT x
TRACE x
NEWLINE
DISTFILE x
OMIT_HEADER
RUN_STAT_DBPARTNUM x
```

Detailed descriptions of these options are provided in *Load configuration options for partitioned database environments*.

RESTARTCOUNT
Reserved.

USING *directory*
Reserved.

Examples of loading data from XML documents

Loading data from an XML document

Example 1

The user has constructed a data file with XDS fields to describe the documents that are to be inserted into the table. It might appear like this :

```
1, "<XDS FIL=""file1.xml"" />"
2, "<XDS FIL='file2.xml' OFF='23' LEN='45' />"
```

For the first row, the XML document is identified by the file named `file1.xml`. Note that since the character delimiter is the double quote character, and double quotation marks exist inside the XDS, the double quotation marks contained within the XDS are doubled. For the second row, the XML document is identified by the file named `file2.xml`, and starts at byte offset 23, and is 45 bytes in length.

Example 2

The user issues a load command without any parsing or validation options for the XML column, and the data is loaded successfully:

```
LOAD FROM data.del of DEL INSERT INTO mytable
```

Loading XML data from CURSOR

Loading data from cursor is the same as with a regular relational column type. The user has two tables, T1 and T2, each of which consist of a single XML column named C1. To LOAD from T1 into T2, the user will first declare a cursor:

```
DECLARE X1 CURSOR FOR SELECT C1 FROM T1;
```

Next, the user may issue a LOAD using the cursor type :

```
LOAD FROM X1 of CURSOR INSERT INTO T2
```

Applying the XML specific LOAD options to the cursor type is the same as loading from a file.

Usage notes

- Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted. If preservation of the source data order is not required, consider using the ANYORDER file type modifier, described below in the *File type modifiers for the load utility* section.
- The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update materialized query tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in Set Integrity Pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in Set Integrity Pending state. Issue the SET INTEGRITY statement to take the tables out of Set Integrity Pending state. Load operations cannot be carried out on replicated materialized query tables.
- If a clustering index exists on the table, the data should be sorted on the clustering index prior to loading. Data does not need to be sorted prior to loading into a multidimensional clustering (MDC) table, however.
- If you specify an exception table when loading into a protected table, any rows that are protected by invalid security labels will be sent to that table. This might allow users that have access to the exception table to access to data that they would not normally be authorized to access. For better security be careful who you grant exception table access to, delete each row as soon as it is repaired and copied to the table being loaded, and drop the exception table as soon as you are done with it.
- Security labels in their internal format might contain newline characters. If you load the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the delprioritychar file type modifier in the LOAD command.
- For performing a load using the `CURSOR` file type where the `DATABASE` keyword was specified during the DECLARE CURSOR command, the user ID and password used to authenticate against the database currently connected to (for the load) will be used to authenticate against the source database (specified by the `DATABASE` option of the DECLARE CURSOR command). If no user ID or password was specified for the connection to the loading database, a user ID and password for the source database must be specified during the DECLARE CURSOR command.
- Loading a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. The names of all the files must be specified in the LOAD command. For example, `LOAD FROM DATA.IXF, DATA.002 OF IXF INSERT INTO TABLE1`. Loading to the Windows operating system from logically split PC/IXF files is not supported.
- When restarting a failed LOAD, the behavior will follow the existing behavior in that the BUILD phase will be forced to use the REBUILD mode for indexes.

Summary of LOAD TERMINATE and LOAD RESTART dictionary management

The following chart summarizes the compression dictionary management behavior for LOAD processing under the TERMINATE directive.

Table 29. LOAD TERMINATE dictionary management

Table COMPRESS Attribute	Does Dictionary exist prior to LOAD?	TERMINATE: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	TERMINATE: LOAD REPLACE RESETDICTIONARY
YES	YES	Keep existing dictionary.	Nothing kept.
YES	NO	Nothing kept.	Nothing kept.
NO	YES	Keep existing dictionary.	Nothing kept.
NO	NO	Do nothing.	Do nothing.

LOAD RESTART truncates a table up to the last consistency point reached. As part of LOAD RESTART processing, a compression dictionary will exist in the table if it was present in the table at the time the last LOAD consistency point was taken. In that case, LOAD RESTART will not create a new dictionary. For a summary of the possible conditions, see Table 4 below.

Table 30. LOAD RESTART dictionary management

Table COMPRESS Attribute	Does Dictionary exist prior to LOAD Consistency point?	RESTART: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	RESTART: LOAD REPLACE RESETDICTIONARY
YES	YES	Keep existing dictionary.	Keep existing dictionary.
YES	NO	Build dictionary subject to ADC.	Build dictionary.
NO	YES	Keep existing dictionary.	Remove existing dictionary.
NO	NO	Do nothing.	Do nothing.

File type modifiers for the load utility

Table 31. Valid file type modifiers for the load utility: All file formats

Modifier	Description
anyorder	This modifier is used in conjunction with the *cpu_parallelism* parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of *cpu_parallelism* is 1, this option is ignored. This option is not supported if SAVECOUNT > 0, since crash recovery after a consistency point requires that data be loaded in sequence.
generatedignore	This modifier informs the load utility that data for all generated columns is present in the data file but should be ignored. This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedmissing` or the `generatedoverride` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated column (not even NULLs). This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedignore` or the `generatedoverride` modifier.

Table 31. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description

generatedoverride	This modifier instructs the load utility to accept user-supplied data for all generated columns in the table (contrary to the normal rules for these types of columns). This is useful when migrating data from another database system, or when loading a table from data that was recovered using the RECOVER DROPPED TABLE option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for a non-nullable generated column will be rejected (SQL3116W). When this modifier is used, the table will be placed in Set Integrity Pending state. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > GENERATED COLUMN IMMEDIATE UNCHECKED`
	
	To take the table out of Set Integrity Pending state and force verification of the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > IMMEDIATE CHECKED.`
	
	When this modifier is specified and there is a generated column in any of the partitioning keys, dimension keys or distribution keys, then the LOAD command will automatically convert the modifier to `generatedignore` and proceed with the load. This will have the effect of regenerating all of the generated column values.
	
	This modifier cannot be used with either the `generatedmissing` or the `generatedignore` modifier.

identityignore	This modifier informs the load utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the `identitymissing` or the `identityoverride` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with either the `identityignore` or the `identityoverride` modifier.
identityoverride	This modifier should be used only when an identity column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of identity columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the identity column will be rejected (SQL3116W). This modifier cannot be used with either the `identitymissing` or the `identityignore` modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's identity column when this option is used.
indexfreespace=*x*	*x* is an integer between 0 and 99 inclusive. The value is interpreted as the percentage of each index page that is to be left as free space when load rebuilds the index. Load with INDEXING MODE INCREMENTAL ignores this option. The first entry in a page is added without restriction; subsequent entries are added to maintain the percent free space threshold. The default value is the one used at CREATE INDEX time. This value takes precedence over the PCTFREE value specified in the CREATE INDEX statement. The indexfreespace option affects index leaf pages only.

Table 31. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column. This option is not supported in conjunction with the CURSOR filetype. The LOBS FROM clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The LOBS FROM clause will implicitly activate the LOBSINFILE behavior. The LOBS FROM clause conveys to the LOAD utility the list of paths to search for the LOB files while loading the data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
noheader	Skips the header verification code (applicable only to load operations into tables that reside in a single-partition database partition group). If the default MPP load (mode PARTITION_AND_LOAD) is used against a table residing in a single-partition database partition group, the file is not expected to have a header. Thus the noheader modifier is not needed. If the LOAD_ONLY mode is used, the file is expected to have a header. The only circumstance in which you should need to use the noheader modifier is if you wanted to perform LOAD_ONLY operation using a file that does not have a header.
norowwarnings	Suppresses all warnings about rejected rows.
pagefreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of each data page that is to be left as free space. If the specified value is invalid because of the minimum row size, (for example, a row that is at least 3 000 bytes long, and an *x* value of 50), the row will be placed on a new page. If a value of 100 is specified, each row will reside on a new page. The PCTFREE value of a table determines the amount of free space designated per page. If a pagefreespace value on the load operation or a PCTFREE value on a table have not been set, the utility will fill up as much space as possible on each page. The value set by pagefreespace overrides the PCTFREE value specified for the table.
rowchangetimestampignore	This modifier informs the load utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMPs being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the `rowchangetimestampmissing` or the `rowchangetimestampoverride` modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with either the `rowchangetimestampignore` or the `rowchangetimestampoverride` modifier.

Table 31. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
rowchangetimestampoverride	This modifier should be used only when a row change timestamp column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of row change timestamp columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the ROW CHANGE TIMESTAMP column will be rejected (SQL3116W). This modifier cannot be used with either the `rowchangetimestampmissing` or the `rowchangetimestampignore` modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's row change timestamp column when this option is used.

seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. LOAD converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3242W) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W) is returned.
	
	This modifier cannot be specified if the seclabelname modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might look like this:
	`"CONFIDENTIAL:ALPHA:G2"`
	`"CONFIDENTIAL;SIGMA:G2"`
	`"TOP SECRET:ALPHA:G2"`
	
	To load or import this data, the SECLABELCHAR file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELCHAR INSERT INTO t1`

seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. LOAD will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned.
	
	This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might consist of security label names similar to:
	`"LABEL1"`
	`"LABEL1"`
	`"LABEL2"`
	
	To load or import this data, the `SECLABELNAME` file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELNAME INSERT INTO t1`
	
	Note: If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.

Table 31. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
totalfreespace=*x*	*x* is an integer greater than or equal to 0 . The value is interpreted as the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if *x* is 20, and the table has 100 data pages after the data has been loaded, 20 additional empty pages will be appended. The total number of data pages for the table will be 120. The data pages total does not factor in the number of index pages in the table. This option does not affect the index object. If two loads are done with this option specified, the second load will not reuse the extra space appended to the end by the first load.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 32. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data (and numeric data specified in characters) from this code page to the database code page during the load operation. The following rules apply: • For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • For DEL data specified in an EBCDIC code page, the delimiters might not coincide with the shift-in and shift-out DBCS characters. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. EBCDIC data can use the corresponding symbols, even though the code points will be different. This option is not supported in conjunction with the CURSOR filetype.

Table 32. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

dateformat="*x*"	*x* is the format of the date in the source file.[1] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

dumpfile = *x*	*x* is the fully qualified (according to the server database partition) name of an exception file to which rejected rows are written. A maximum of 32 KB of data is written per record. Following is an example that shows how to specify a dump file:
	`db2 load from data of del`
	`modified by dumpfile = /u/user/filename`
	`insert into table_name`
	The file will be created and owned by the instance owner. To override the default file permissions, use the `dumpfileaccessall` file type modifier.
	Note:
	1. In a partitioned database environment, the path should be local to the loading database partition, so that concurrently running load operations do not attempt to write to the same file.
	2. The contents of the file are written to disk in an asynchronous buffered mode. In the event of a failed or an interrupted load operation, the number of records committed to disk cannot be known with certainty, and consistency cannot be guaranteed after a LOAD RESTART. The file can only be assumed to be complete for a load operation that starts and completes in a single pass.
	3. If the specified file already exists, it will not be recreated, but it will be appended.

dumpfileaccessall	Grants read access to 'OTHERS' when a dump file is created. This file type modifier is only valid when: 1. it is used in conjunction with `dumpfile` file type modifier 2. the user has SELECT privilege on the load target table 3. it is issued on a DB2 server database partition that resides on a UNIX operating system If the specified file already exists, its permissions will not be changed.
fastparse	Use with caution. Reduces syntax checking on user-supplied column values, and enhances performance. Tables are guaranteed to be architecturally correct (the utility performs sufficient data checking to prevent a segmentation violation or trap), however, the coherence of the data is not validated. Only use this option if you are certain that your data is coherent and correct. For example, if the user-supplied data contains an invalid timestamp column value of `:1>0-00-20-07.11.12.000000`, this value is inserted into the table if FASTPARSE is specified, and rejected if FASTPARSE is not specified.

Table 32. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`. This modifier cannot be used with the `packeddecimal` modifier.

timeformat="*x*"	*x* is the format of the time in the source file.[1] Valid time elements are:
	`H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M - Minute (one or two digits ranging from 0 - 59)`
	`MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M)`
	`S - Second (one or two digits ranging from 0 - 59)`
	`SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`TT - Meridian indicator (AM or PM)`
	
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 32. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
timestampformat="*x*"	*x* is the format of the time stamp in the source file.[1] Valid time stamp elements are: YYYY - Year (four digits ranging from 0000 - 9999) M - Month (one or two digits ranging from 1 - 12) MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM) MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM) D - Day (one or two digits ranging from 1 - 31) DD - Day (two digits ranging from 1 - 31; mutually exclusive with D) DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements) H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system) HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H) M - Minute (one or two digits ranging from 0 - 59) MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute) S - Second (one or two digits ranging from 0 - 59) SS - Second (two digits ranging from 0 - 59; mutually exclusive with S) SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements) UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements) UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microseond elements) UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microseond elements) UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microseond elements) UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microseond elements) U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microseond elements) TT - Meridian indicator (AM or PM)

Table 32. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*" (Continued)	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	If the TIMESTAMPFORMAT modifier is not specified, the load utility formats the timestamp field using one of two possible formats:
	`YYYY-MM-DD-HH.MM.SS`
	`YYYY-MM-DD HH:MM:SS`
	The load utility chooses the format by looking at the separator between the DD and HH. If it is a dash '-', the load utility uses the regular dashes and dots format (YYYY-MM-DD-HH.MM.SS). If it is a blank space, then the load utility expects a colon ':' to separate the HH, MM and SS.
	In either format, if you include the microseconds field (UUUUUU), the load utility expects the dot '.' as the separator. Either YYYY-MM-DD-HH.MM.SS.UUUUUU or YYYY-MM-DD HH:MM:SS.UUUUUU are acceptable.
	The following example illustrates how to load data containing user defined date and time formats into a table called schedule:
	`db2 load from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being loaded into graphic or double-byte character large object (DBCLOB) data field(s) is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic codepage is associated with the character code page. LOAD determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the database if the `codepage` modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The `usegraphiccodepage` modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The `usegraphiccodepage` modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.

Table 32. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the `codepage` file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute.

Table 33. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII)

Modifier	Description
binarynumerics	Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions. This option is supported only with positional ASC, using fixed length records specified by the `reclen` option. The following rules apply: • No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT. • Data lengths must match their target column definitions. • FLOATs must be in IEEE Floating Point format. • Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running. NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a NULL value to *x*. The default value of *x* is `Y`.[2] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the NULL indicator character is specified to be the letter `N`, then `n` is also recognized as a NULL indicator.

Table 33. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII) (continued)

Modifier	Description

packeddecimal	Loads packed-decimal data directly, since the `binarynumerics` modifier does not include the DECIMAL field type.
	
	This option is supported only with positional ASC, using fixed length records specified by the `reclen` option.
	
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
	
	Regardless of the server platform, the byte order of binary data in the load source file is assumed to be big-endian; that is, when using this modifier on Windows operating systems, the byte order must not be reversed.
	
	This modifier cannot be used with the `implieddecimal` modifier.

reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.
striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept. This option cannot be specified together with `striptnulls`. These are mutually exclusive options. This option replaces the obsolete `t` option, which is supported for earlier compatibility only.
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. This option replaces the obsolete `padwithzero` option, which is supported for earlier compatibility only.

zoneddecimal	Loads zoned decimal data, since the BINARYNUMERICS modifier does not include the DECIMAL field type. This option is supported only with positional ASC, using fixed length records specified by the RECLEN option.
	
	Half-byte sign values can be one of the following:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	Supported values for digits are `0x0` to `0x9`.
	
	Supported values for zones are `0x3` and `0xF`.

Table 34. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII)

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[23] If you want to explicitly specify the double quotation mark (″) as the character string delimiter, you should specify it as follows:
	`modified by chardel""`
	The single quotation mark (′) can also be specified as a character string delimiter as follows:
	`modified by chardel''`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[23]
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[23]

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 load ... modified by delprioritychar ...`
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
	The following example illustrates how to load data into a table called TABLE1, while preserving all leading and trailing spaces in the data file:
	`db2 load from delfile3 of del`
	`modified by keepblanks`
	`insert into table1`

nochardel	The load utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx`, `delprioritychar` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 35. Valid file type modifiers for the load utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to load each row.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Note:

1. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

   For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```

   Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
2. Character values provided for the chardel, coldel, or decpt file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
3. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.

4. The load utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the load operation fails, and an error code is returned.
5. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the rowchangetimestampmissing file type modifier *must be* specified in the import command if data for the column is not present in the data to be imported and there is no explicit column list present.

Table 36. LOAD behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Absent	All data in the file is assumed to be in the database code page, not the application code page, even if the CLIENT option is specified.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the database code page, even if the CLIENT option is specified. Graphic data is assumed to be in the code page of the database graphic data, even if the CLIENT option is specified. If the database code page is single-byte, then all data is assumed to be in the database code page. **Warning:** Graphic data will be corrupted when loaded into a single-byte database.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.

LOAD command using the ADMIN_CMD procedure

Loads data into a DB2 table. Data residing on the server can be in the form of a file, tape, or named pipe. If the `COMPRESS` attribute for the table is set to `YES`, the data loaded will be subject to compression on every data and database partition for which a dictionary already exists in the table.

Quick link to "File type modifiers for the load utility" on page 248.

Restrictions

The load utility does not support loading data at the hierarchy level. The load utility is not compatible with range-clustered tables.

Scope

This command can be issued against multiple database partitions in a single request.

Authorization

One of the following:

- *sysadm*
- *dbadm*
- LOAD authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the load fails and an error (SQLSTATE 5U014) is returned.
- To load data into a table that has protected rows, the session authorization id must hold a security label that meets these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access or for all access

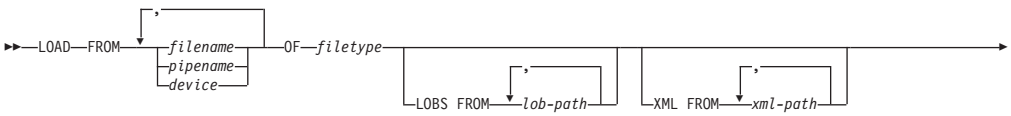
  If the session authorization id does not hold such a security label then the load fails and an error (SQLSTATE 5U014) is returned. This security label is used to protect a loaded row if the session authorization ID's LBAC credentials do not allow it to write to the security label that protects that row in the data. This does not happen, however, when the security policy protecting the table was created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option of the CREATE SECURITY POLICY statement. In this case the load fails and an error (SQLSTATE 42519) is returned.
- If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.

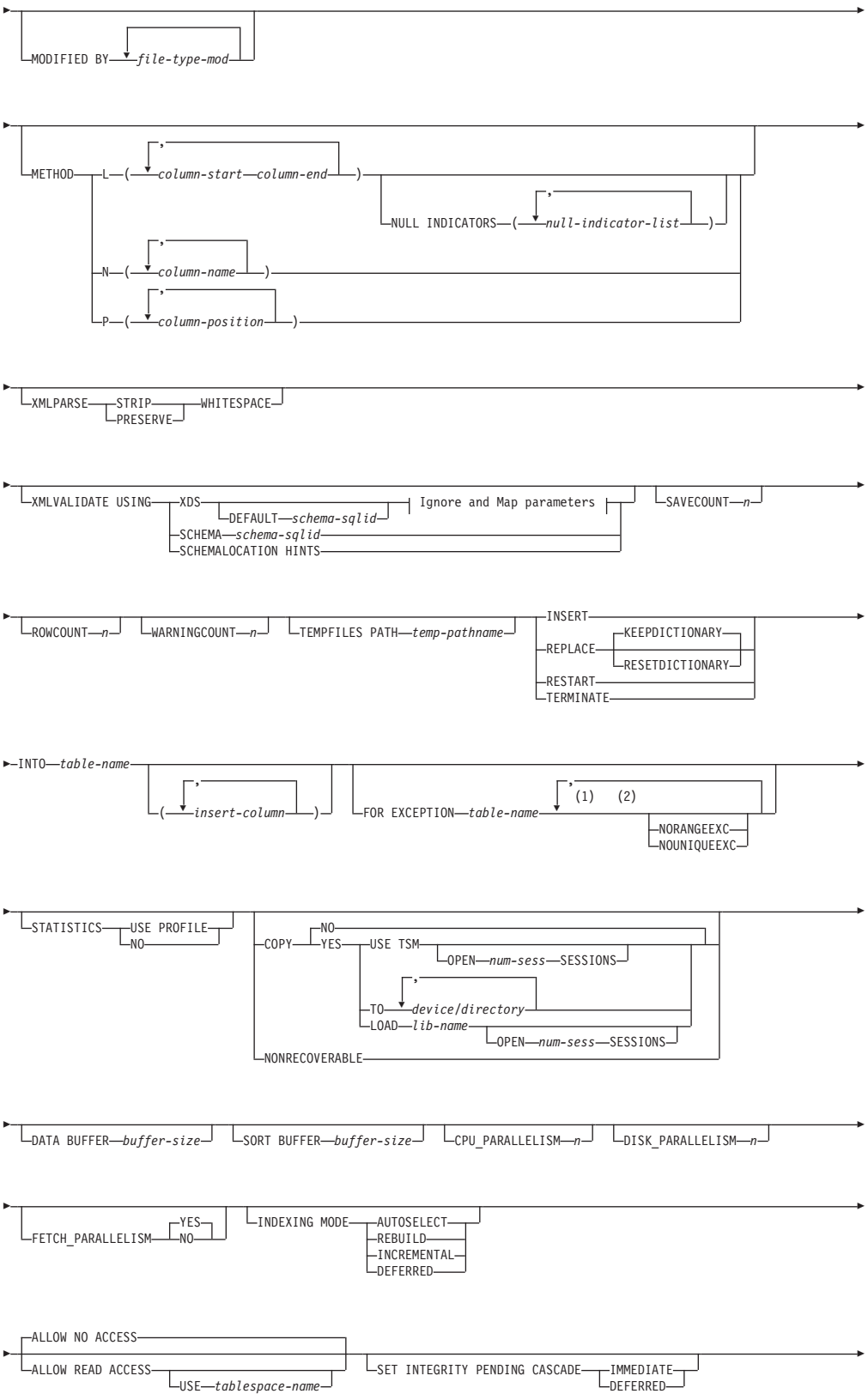
Since all load processes (and all DB2 server processes, in general) are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

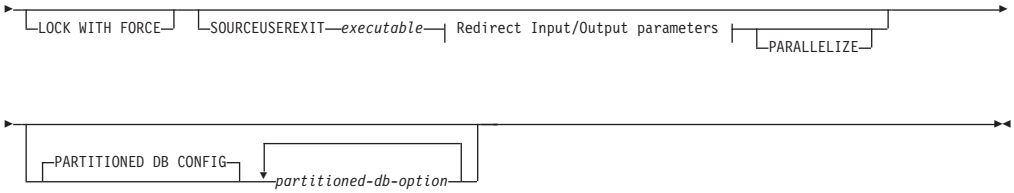
Required connection

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

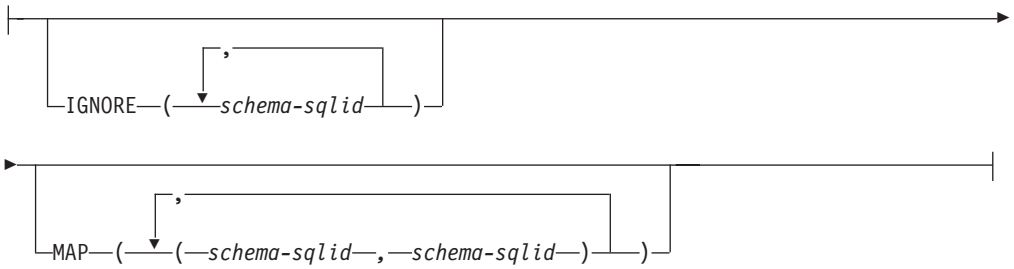
Command syntax



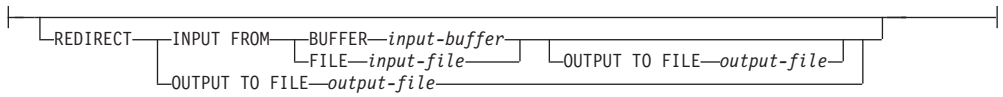

MODIFIED BY
file-type-mod
METHOD
L
column-start
column-end
NULL INDICATORS
null-indicator-list
N
column-name
P
column-position
XMLPARSE
STRIP
PRESERVE
WHITESPACE
XMLVALIDATE USING
XDS
DEFAULT
schema-sqlid
SCHEMA
schema-sqlid
SCHEMALOCATION HINTS
Ignore and Map parameters
SAVECOUNT
n
ROWCOUNT
n
WARNINGCOUNT
n
TEMPFILES PATH
temp-pathname
INSERT
REPLACE
KEEPDICTIONARY
RESETDICTIONARY
RESTART
TERMINATE
INTO
table-name
insert-column
FOR EXCEPTION
table-name
(1) (2)
NORANGEEXC
NOUNIQUEEXC
STATISTICS
USE PROFILE
NO
COPY
NO
YES
USE TSM
OPEN
num-sess
SESSIONS
TO
device/directory
LOAD
lib-name
OPEN
num-sess
SESSIONS
NONRECOVERABLE
DATA BUFFER
buffer-size
SORT BUFFER
buffer-size
CPU_PARALLELISM
n
DISK_PARALLELISM
n
FETCH_PARALLELISM
YES
NO
INDEXING MODE
AUTOSELECT
REBUILD
INCREMENTAL
DEFERRED
ALLOW NO ACCESS
ALLOW READ ACCESS
USE
tablespace-name
SET INTEGRITY PENDING CASCADE
IMMEDIATE
DEFERRED



Ignore and Map parameters:



Redirect Input/Output parameters:



Notes:

1 These keywords can appear in any order.

2 Each of these keywords can only appear once.

Command parameters

FROM *filename* | *pipename* | *device*

Note:

1. If data is exported into a file using the *EXPORT command using the ADMIN_CMD procedure*, the data file is owned by the fenced user ID. This file is not usually accessible by the instance owner. To run the LOAD from CLP or the ADMIN_CMD procedure, the data file must be accessible by the instance owner ID, so read access to the data file must be granted to the instance owner.
2. Loading data from multiple IXF files is supported if the files are physically separate, but logically one file. It is *not* supported if the files are both logically and physically separate. (Multiple physical files would be considered logically one if they were all created with one invocation of the EXPORT command.)

OF *filetype*

Specifies the format of the data:

- ASC (non-delimited ASCII format).
- DEL (delimited ASCII format).
- IXF (Integration Exchange Format, PC version) is a binary format that is used exclusively by DB2.

- CURSOR (a cursor declared against a SELECT or VALUES statement).

LOBS FROM *lob-path*
The path to the data files containing LOB values to be loaded. The path must end with a slash (/). The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the LOBSINFILE behavior.

This option is ignored when specified in conjunction with the CURSOR file type.

MODIFIED BY *file-type-mod*
Specifies file type modifier options. See "File type modifiers for the load utility" on page 248.

METHOD

L Specifies the start and end column numbers from which to load data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1. This method can only be used with ASC files, and is the only valid method for that file type.

NULL INDICATORS *null-indicator-list*
This option can only be used when the METHOD L parameter is specified; that is, the input file is an ASC file). The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the METHOD L parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of Y in the NULL indicator column specifies that the column data is NULL. Any character *other than* Y in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the METHOD L option will be loaded.

The NULL indicator character can be changed using the MODIFIED BY option.

N Specifies the names of the columns in the data file to be loaded. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the METHOD N list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method N (F2, F1, F4, F3)` is a valid request, while `method N (F2, F1)` is not valid. This method can only be used with file types IXF or CURSOR.

P Specifies the field numbers (numbered from 1) of the input data fields to be loaded. Each table column that is not nullable should have a corresponding entry in the METHOD P list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method P (2, 1, 4, 3)` is a valid request, while `method P (2, 1)` is not

valid. This method can only be used with file types IXF, DEL, or CURSOR, and is the only valid method for the DEL file type.

XML FROM *xml-path*
Specifies one or more paths that contain the XML files. XDSs are contained in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the XML column.

XMLPARSE
Specifies how XML documents are parsed. If this option is not specified, the parsing behavior for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

STRIP WHITESPACE
Specifies to remove whitespace when the XML document is parsed.

PRESERVE WHITESPACE
Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the XMLVALIDATE option is invoked with the USING XDS clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the DEFAULT clause.

The DEFAULT, IGNORE, and MAP clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the DEFAULT clause, it will not be ignored if also specified by the IGNORE clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the MAP clause, it will not be re-mapped if also specified in the second part of another MAP clause pair.

USING SCHEMA *schema-sqlid*
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation attribute is not found in the XML document, no validation will occur. When the USING SCHEMALOCATION HINTS clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the XMLVALIDATE option below.

IGNORE *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The IGNORE clause specifies a list of one or more schemas to ignore if

they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for a loaded XML document, and the schema identified by the SCH attribute is included in the list of schemas to IGNORE, then no schema validation will occur for the loaded XML document.

Note:

If a schema is specified in the IGNORE clause, it cannot also be present in the left side of a schema pair in the MAP clause.

The IGNORE clause applies only to the XDS. A schema that is mapped by the MAP clause will not be subsequently ignored if specified by the IGNORE clause.

DEFAULT *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. The schema specified through the DEFAULT clause identifies a schema to use for validation when the XML Data Specifier (XDS) of a loaded XML document does not contain an SCH attribute identifying an XML Schema.

The DEFAULT clause takes precedence over the IGNORE and MAP clauses. If an XDS satisfies the DEFAULT clause, the IGNORE and MAP specifications will be ignored.

MAP *schema-sqlid*
This option can only be used when the USING XDS parameter is specified. Use the MAP clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each loaded XML document. The MAP clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the MAP clause, it cannot also be specified in the IGNORE clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

SAVECOUNT *n*
Specifies that the load utility is to establish consistency points after every *n* rows. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using LOAD QUERY. If the value of *n* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is zero, meaning that no consistency points will be established, unless necessary.

This option is ignored when specified in conjunction with the CURSOR file type.

ROWCOUNT *n*

Specifies the number of *n* physical records in the file to be loaded. Allows a user to load only the first *n* rows in a file.

WARNINGCOUNT *n*

Stops the load operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If *n* is zero, or this option is not specified, the load operation will continue regardless of the number of warnings issued. If the load operation is stopped because the threshold of warnings was encountered, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

TEMPFILES PATH *temp-pathname*

Specifies the name of the path to be used when creating temporary files during a load operation, and should be fully qualified according to the server database partition.

Temporary files take up file system space. Sometimes, this space requirement is quite substantial. Following is an estimate of how much file system space should be allocated for all temporary files:

- 136 bytes for each message that the load utility generates
- 15 KB overhead if the data file contains long field data or LOBs. This quantity can grow significantly if the INSERT option is specified, and there is a large amount of long field or LOB data already in the table.

INSERT

One of four modes under which the load utility can execute. Adds the loaded data to the table without changing the existing table data.

REPLACE

One of four modes under which the load utility can execute. Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

KEEPDICTIONARY

An existing compression dictionary is preserved across the LOAD REPLACE operation. Provided the table COMPRESS attribute is YES, the newly replaced data is subject to being compressed using the dictionary that existed prior to the invocation of the load. If no dictionary previously existed in the table, a new dictionary is built using the data that is being replaced into the table as long as the table COMPRESS attribute is YES. The amount of data that is required to build the compression dictionary in this case is subject to the policies of ADC. This data is populated into the table as uncompressed. Once the dictionary is inserted into the table, the remaining data to be loaded is subject to being compressed with this dictionary. This is the default parameter. For summary, see Table 1 below.

Table 37. LOAD REPLACE KEEPDICTIONARY

Compress	Dictionary exists	Result outcome
Y	Y	Preserve dictionary; all input rows are subject to compression with existing dictionary.
Y	N	Insert new dictionary into table only if sufficient user data exists; remaining rows subject to compression after dictionary built.
N	Y	Preserve dictionary; all input rows are not compressed.
N	N	No effect; all rows not compressed.

RESETDICTIONARY

This directive instructs LOAD REPLACE processing to build a new dictionary for the table data object provided that the table COMPRESS attribute is YES. If the COMPRESS attribute is NO and a dictionary was already present in the table it will be removed and no new dictionary will be inserted into the table. A compression dictionary can be built with just one user record. If the loaded data set size is zero and if there is a pre-existing dictionary, the dictionary will not be preserved. The amount of data required to build a dictionary with this directive is not subject to the policies of ADC. For summary, see Table 2 below.

Table 38. LOAD REPLACE RESETDICTIONARY

Compress	Dictionary exists	Result outcome
Y	Y	Build a new dictionary*; remaining rows to be loaded are subject to compression after dictionary built.
Y	N	Build new dictionary; remaining rows subject to compression after dictionary built.
N	Y	Remove dictionary; all input rows are not compressed.
N	N	No effect; all rows not compressed.

* If a dictionary exists and the compression attribute is enabled, but there are no records to load into the table partition, a new dictionary cannot be built and the RESETDICTIONARY operation will not keep the existing dictionary.

TERMINATE

One of four modes under which the load utility can execute. Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects might be marked as invalid, in which case index rebuild will automatically take place at next access). If the load operation being terminated is a LOAD REPLACE, the table will be truncated to an empty table after the LOAD TERMINATE operation. If the load operation being terminated is a LOAD INSERT, the table will retain all of its original records after the LOAD TERMINATE operation. For summary of dictionary management, see Table 3 below.

The LOAD TERMINATE option will not remove a backup pending state from table spaces.

RESTART
One of four modes under which the load utility can execute. Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase. For summary of dictionary management, see Table 4 below.

INTO *table-name*
Specifies the database table into which the data is to be loaded. This table cannot be a system table or a declared temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form `schema.tablename`. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

insert-column
Specifies the table column into which the data is to be inserted.

The load utility cannot parse columns whose names contain one or more spaces. For example,

will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

FOR EXCEPTION *table-name*
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

Information that is written to the exception table is *not* written to the dump file. In a partitioned database environment, an exception table must be defined for those database partitions on which the loading table is defined. The dump file, otherwise, contains rows that cannot be loaded because they are invalid or have syntax errors.

NORANGEEXC
Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

NOUNIQUEEXC
Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

STATISTICS USE PROFILE
Instructs load to collect statistics during the load according to the profile defined for this table. This profile must be created before load is executed. The profile is created by the RUNSTATS command. If the profile does not exist and load is instructed to collect statistics according to the profile, a warning is returned and no statistics are collected.

STATISTICS NO
Specifies that no statistics are to be collected, and that the statistics in the catalogs are not to be altered. This is the default.

COPY NO
Specifies that the table space in which the table resides will be placed in backup pending state if forward recovery is enabled (that is, *logretain* or *userexit* is on). The COPY NO option will also put the table space state into the Load in Progress table space state. This is a transient state that will disappear when the load completes or aborts. The data in any table in the table space cannot be updated or deleted until a table space backup or a

full database backup is made. However, it is possible to access the data in any table by using the SELECT statement.

LOAD with COPY NO on a recoverable database leaves the table spaces in a backup pending state. For example, performing a LOAD with COPY NO and INDEXING MODE DEFERRED will leave indexes needing a refresh. Certain queries on the table might require an index scan and will not succeed until the indexes are refreshed. The index cannot be refreshed if it resides in a table space which is in the backup pending state. In that case, access to the table will not be allowed until a backup is taken. Index refresh is done automatically by the database when the index is accessed by a query. If one of COPY NO, COPY YES, or NONRECOVERABLE is not specified, and the database is recoverable (**logretain** or **logarchmeth1** is enabled), then COPY NO is the default.

COPY YES
Specifies that a copy of the loaded data will be saved. This option is invalid if forward recovery is disabled (both *logretain* and *userexit* are off).

USE TSM
Specifies that the copy will be stored using Tivoli Storage Manager (TSM).

OPEN *num-sess* **SESSIONS**
The number of I/O sessions to be used with TSM or the vendor product. The default value is 1.

TO *device/directory*
Specifies the device or directory on which the copy image will be created.

LOAD *lib-name*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path where the user exit programs reside.

NONRECOVERABLE
Specifies that the load transaction is to be marked as non-recoverable and that it will not be possible to recover it by a subsequent roll forward action. The roll forward utility will skip the transaction and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward operation is completed, such a table can only be dropped or restored from a backup (full or table space) taken after a commit point following the completion of the non-recoverable load operation.

With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. If one of COPY NO, COPY YES, or NONRECOVERABLE is not specified, and the database is not recoverable (**logretain** or **logarchmeth1** is not enabled), then NONRECOVERABLE is the default.

WITHOUT PROMPTING
Specifies that the list of data files contains all the files that are to be loaded, and that the devices or directories listed are sufficient for the entire load operation. If a continuation input file is not found, or the copy targets are filled before the load operation finishes, the load operation will fail, and the table will remain in load pending state.

DATA BUFFER *buffer-size*
Specifies the number of 4 KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the minimum required resource is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the *util_heap_sz* database configuration parameter.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

SORT BUFFER *buffer-size*
This option specifies a value that overrides the SORTHEAP database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the INDEXING MODE parameter is not specified as DEFERRED. The value that is specified cannot exceed the value of SORTHEAP. This parameter is useful for throttling the sort memory that is used when loading tables with many indexes without changing the value of SORTHEAP, which would also affect general query processing.

CPU_PARALLELISM *n*
Specifies the number of processes or threads that the load utility will create for parsing, converting, and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, or has not been specified, the load utility uses an intelligent default value (usually based on the number of CPUs available) at run time.

Note:

1. If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs or the value specified by the user.
2. Specifying a small value for the SAVECOUNT parameter causes the loader to perform many more I/O operations to flush both data and table metadata. When CPU_PARALLELISM is greater than one, the flushing operations are asynchronous, permitting the loader to exploit the CPU. When CPU_PARALLELISM is set to one, the loader waits on I/O during consistency points. A load operation with CPU_PARALLELISM set to two, and SAVECOUNT set to 10 000, completes faster than the same operation with CPU_PARALLELISM set to one, even though there is only one CPU.

DISK_PARALLELISM *n*
Specifies the number of processes or threads that the load utility will create for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

FETCH_PARALLELISM YES | NO
When performing a load from a cursor where the cursor is declared using the DATABASE keyword, or when using the API `sqlu_remotefetch_entry` media entry, and this option is set to YES, the load utility attempts to parallelize fetching from the remote data source if possible. If set to NO,

no parallel fetching is performed. The default value is YES. For more information, see *Moving data using the CURSOR file type*.

INDEXING MODE

Specifies whether the load utility is to rebuild indexes or to extend them incrementally. Valid values are:

AUTOSELECT

The load utility will automatically decide between REBUILD or INCREMENTAL mode. The decision is based on the amount of data being loaded and the depth of the index tree. Information relating to the depth of the index tree is stored in the index object. RUNSTATS is not required to populate this information. AUTOSELECT is the default indexing mode.

REBUILD

All indexes will be rebuilt. The utility must have sufficient resources to sort all index key parts for both old and appended table data.

INCREMENTAL

Indexes will be extended with new data. This approach consumes index free space. It only requires enough sort space to append index keys for the inserted records. This method is only supported in cases where the index object is valid and accessible at the start of a load operation (it is, for example, not valid immediately following a load operation in which the DEFERRED mode was specified). If this mode is specified, but not supported due to the state of the index, a warning is returned, and the load operation continues in REBUILD mode. Similarly, if a load restart operation is begun in the load build phase, INCREMENTAL mode is not supported.

Incremental indexing is not supported when all of the following conditions are true:

- The LOAD COPY option is specified (*logarchmeth1* with the `USEREXIT` or `LOGRETAIN` option).
- The table resides in a DMS table space.
- The index object resides in a table space that is shared by other table objects belonging to the table being loaded.

To bypass this restriction, it is recommended that indexes be placed in a separate table space.

DEFERRED

The load utility will not attempt index creation if this mode is specified. Indexes will be marked as needing a refresh. The first access to such indexes that is unrelated to a load operation might force a rebuild, or indexes might be rebuilt when the database is restarted. This approach requires enough sort space for all key parts for the largest index. The total time subsequently taken for index construction is longer than that required in REBUILD mode. Therefore, when performing multiple load operations with deferred indexing, it is advisable (from a performance viewpoint) to let the last load operation in the sequence perform an index rebuild, rather than allow indexes to be rebuilt at first non-load access.

Deferred indexing is only supported for tables with non-unique indexes, so that duplicate keys inserted during the load phase are not persistent after the load operation.

ALLOW NO ACCESS
Load will lock the target table for exclusive access during the load. The table state will be set to Load In Progress during the load. ALLOW NO ACCESS is the default behavior. It is the only valid option for LOAD REPLACE.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending state.

ALLOW READ ACCESS
Load will lock the target table in a share mode. The table state will be set to both Load In Progress and Read Access. Readers can access the non-delta portion of the data while the table is being load. In other words, data that existed before the start of the load will be accessible by readers to the table, data that is being loaded is not available until the load is complete. LOAD TERMINATE or LOAD RESTART of an ALLOW READ ACCESS load can use this option; LOAD TERMINATE or LOAD RESTART of an ALLOW NO ACCESS load cannot use this option. Furthermore, this option is not valid if the indexes on the target table are marked as requiring a rebuild.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress, and Read Access. At the end of the load, the table state Load In Progress will be removed but the table states Set Integrity Pending and Read Access will remain. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending. While the table is in Set Integrity Pending and Read Access states, the non-delta portion of the data is still accessible to readers, the new (delta) portion of the data will remain inaccessible until the SET INTEGRITY statement has completed. A user can perform multiple loads on the same table without issuing a SET INTEGRITY statement. Only the original (checked) data will remain visible, however, until the SET INTEGRITY statement is issued.

ALLOW READ ACCESS also supports the following modifiers:

USE *tablespace-name*
If the indexes are being rebuilt, a shadow copy of the index is built in table space *tablespace-name* and copied over to the original table space at the end of the load during an INDEX COPY PHASE. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same table space as the index object. If the shadow copy is created in the same table space as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different table space from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load during the INDEX COPY PHASE.

Without this option the shadow index is built in the same table space as the original. Since both the original index and shadow index by default reside in the same table space simultaneously,

there might be insufficient space to hold both indexes within one table space. Using this option ensures that you retain enough table space for the indexes.

This option is ignored if the user does not specify INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT. This option will also be ignored if INDEXING MODE AUTOSELECT is chosen and load chooses to incrementally update the index.

SET INTEGRITY PENDING CASCADE

If LOAD puts the table into Set Integrity Pending state, the SET INTEGRITY PENDING CASCADE option allows the user to specify whether or not Set Integrity Pending state of the loaded table is immediately cascaded to all descendents (including descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables).

IMMEDIATE

Indicates that Set Integrity Pending state is immediately extended to all descendent foreign key tables, descendent immediate materialized query tables and descendent staging tables. For a LOAD INSERT operation, Set Integrity Pending state is not extended to descendent foreign key tables even if the IMMEDIATE option is specified.

When the loaded table is later checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement), descendent foreign key tables that were placed in Set Integrity Pending Read Access state will be put into Set Integrity Pending No Access state.

DEFERRED

Indicates that only the loaded table will be placed in the Set Integrity Pending state. The states of the descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged.

Descendent foreign key tables might later be implicitly placed in Set Integrity Pending state when their parent tables are checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement). Descendent immediate materialized query tables and descendent immediate staging tables will be implicitly placed in Set Integrity Pending state when one of its underlying tables is checked for integrity violations. A warning (SQLSTATE 01586) will be issued to indicate that dependent tables have been placed in Set Integrity Pending state. See the Notes section of the SET INTEGRITY statement in the SQL Reference for when these descendent tables will be put into Set Integrity Pending state.

If the SET INTEGRITY PENDING CASCADE option is not specified:

- Only the loaded table will be placed in Set Integrity Pending state. The state of descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged, and can later be implicitly put into Set Integrity Pending state when the loaded table is checked for constraint violations.

If LOAD does not put the target table into Set Integrity Pending state, the SET INTEGRITY PENDING CASCADE option is ignored.

LOCK WITH FORCE
The utility acquires various locks including table locks in the process of loading. Rather than wait, and possibly timeout, when acquiring a lock, this option allows load to force off other applications that hold conflicting locks on the target table. Applications holding conflicting locks on the system catalog tables will not be forced off by the load utility. Forced applications will roll back and release the locks the load utility needs. The load utility can then proceed. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

ALLOW NO ACCESS loads might force applications holding conflicting locks at the start of the load operation. At the start of the load the utility can force applications that are attempting to either query or modify the table.

ALLOW READ ACCESS loads can force applications holding conflicting locks at the start or end of the load operation. At the start of the load the load utility can force applications that are attempting to modify the table. At the end of the load operation, the load utility can force applications that are attempting to either query or modify the table.

SOURCEUSEREXIT*executable*
Specifies an executable filename which will be called to feed data into the utility.

REDIRECT

INPUT FROM

BUFFER *input-buffer*
The stream of bytes specified in *input-buffer* is passed into the STDIN file descriptor of the process executing the given executable.

FILE *input-file*
The contents of this client-side file are passed into the STDIN file descriptor of the process executing the given executable.

OUTPUT TO

FILE *output-file*
The STDOUT and STDERR file descriptors are captured to the fully qualified server-side file specified.

PARALLELIZE
Increases the throughput of data coming into the load utility by invoking multiple user exit processes simultaneously. This option is only applicable in multi-partition database environments and is ignored in single-partition database environments.

For more information, see *Moving data using a customized application (user exit)*.

PARTITIONED DB CONFIG *partitioned-db-option*
Allows you to execute a load into a table distributed across multiple database partitions. The PARTITIONED DB CONFIG parameter allows you to specify partitioned database-specific configuration options. The *partitioned-db-option* values can be any of the following:

```
PART_FILE_LOCATION x
OUTPUT_DBPARTNUMS x
PARTITIONING_DBPARTNUMS x
MODE x
MAX_NUM_PART_AGENTS x
ISOLATE_PART_ERRS x
STATUS_INTERVAL x
PORT_RANGE x
CHECK_TRUNCATION
MAP_FILE_INPUT x
MAP_FILE_OUTPUT x
TRACE x
NEWLINE
DISTFILE x
OMIT_HEADER
RUN_STAT_DBPARTNUM x
```

Detailed descriptions of these options are provided in *Load configuration options for partitioned database environments*.

RESTARTCOUNT
Reserved.

USING *directory*
Reserved.

Examples of loading data from XML documents

Loading data from an XML document

Example 1

The user has constructed a data file with XDS fields to describe the documents that are to be inserted into the table. It might appear like this :

```
1, "<XDS FIL=""file1.xml"" />"
2, "<XDS FIL='file2.xml' OFF='23' LEN='45' />"
```

For the first row, the XML document is identified by the file named `file1.xml`. Note that since the character delimiter is the double quote character, and double quotation marks exist inside the XDS, the double quotation marks contained within the XDS are doubled. For the second row, the XML document is identified by the file named `file2.xml`, and starts at byte offset 23, and is 45 bytes in length.

Example 2

The user issues a load command without any parsing or validation options for the XML column, and the data is loaded successfully:

```
LOAD FROM data.del of DEL INSERT INTO mytable
```

Loading XML data from CURSOR

Loading data from cursor is the same as with a regular relational column type. The user has two tables, T1 and T2, each of which consist of a single XML column named C1. To LOAD from T1 into T2, the user will first declare a cursor:

```
DECLARE X1 CURSOR FOR SELECT C1 FROM T1;
```

Next, the user may issue a LOAD using the cursor type :

```
LOAD FROM X1 of CURSOR INSERT INTO T2
```

Applying the XML specific LOAD options to the cursor type is the same as loading from a file.

Usage notes

- Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted. If preservation of the source data order is not required, consider using the ANYORDER file type modifier, described below in the *File type modifiers for the load utility* section.
- The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update materialized query tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in Set Integrity Pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in Set Integrity Pending state. Issue the SET INTEGRITY statement to take the tables out of Set Integrity Pending state. Load operations cannot be carried out on replicated materialized query tables.
- If a clustering index exists on the table, the data should be sorted on the clustering index prior to loading. Data does not need to be sorted prior to loading into a multidimensional clustering (MDC) table, however.
- If you specify an exception table when loading into a protected table, any rows that are protected by invalid security labels will be sent to that table. This might allow users that have access to the exception table to access to data that they would not normally be authorized to access. For better security be careful who you grant exception table access to, delete each row as soon as it is repaired and copied to the table being loaded, and drop the exception table as soon as you are done with it.
- Security labels in their internal format might contain newline characters. If you load the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the delprioritychar file type modifier in the LOAD command.
- For performing a load using the `CURSOR` file type where the `DATABASE` keyword was specified during the DECLARE CURSOR command, the user ID and password used to authenticate against the database currently connected to (for the load) will be used to authenticate against the source database (specified by the `DATABASE` option of the DECLARE CURSOR command). If no user ID or password was specified for the connection to the loading database, a user ID and password for the source database must be specified during the DECLARE CURSOR command.
- Loading a multiple-part PC/IXF file whose individual parts are copied from a Windows system to an AIX system is supported. The names of all the files must be specified in the LOAD command. For example, `LOAD FROM DATA.IXF, DATA.002 OF IXF INSERT INTO TABLE1`. Loading to the Windows operating system from logically split PC/IXF files is not supported.
- When restarting a failed LOAD, the behavior will follow the existing behavior in that the BUILD phase will be forced to use the REBUILD mode for indexes.

Summary of LOAD TERMINATE and LOAD RESTART dictionary management

The following chart summarizes the compression dictionary management behavior for LOAD processing under the TERMINATE directive.

Table 39. LOAD TERMINATE dictionary management

Table COMPRESS Attribute	Does Dictionary exist prior to LOAD?	TERMINATE: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	TERMINATE: LOAD REPLACE RESETDICTIONARY
YES	YES	Keep existing dictionary.	Nothing kept.
YES	NO	Nothing kept.	Nothing kept.
NO	YES	Keep existing dictionary.	Nothing kept.
NO	NO	Do nothing.	Do nothing.

LOAD RESTART truncates a table up to the last consistency point reached. As part of LOAD RESTART processing, a compression dictionary will exist in the table if it was present in the table at the time the last LOAD consistency point was taken. In that case, LOAD RESTART will not create a new dictionary. For a summary of the possible conditions, see Table 4 below.

Table 40. LOAD RESTART dictionary management

Table COMPRESS Attribute	Does Dictionary exist prior to LOAD Consistency point?	RESTART: LOAD REPLACE KEEPDICTIONARY or LOAD INSERT	RESTART: LOAD REPLACE RESETDICTIONARY
YES	YES	Keep existing dictionary.	Keep existing dictionary.
YES	NO	Build dictionary subject to ADC.	Build dictionary.
NO	YES	Keep existing dictionary.	Remove existing dictionary.
NO	NO	Do nothing.	Do nothing.

File type modifiers for the load utility

Table 41. Valid file type modifiers for the load utility: All file formats

Modifier	Description
anyorder	This modifier is used in conjunction with the *cpu_parallelism* parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of *cpu_parallelism* is 1, this option is ignored. This option is not supported if SAVECOUNT > 0, since crash recovery after a consistency point requires that data be loaded in sequence.
generatedignore	This modifier informs the load utility that data for all generated columns is present in the data file but should be ignored. This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedmissing` or the `generatedoverride` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated column (not even NULLs). This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedignore` or the `generatedoverride` modifier.

Table 41. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description

generatedoverride	This modifier instructs the load utility to accept user-supplied data for all generated columns in the table (contrary to the normal rules for these types of columns). This is useful when migrating data from another database system, or when loading a table from data that was recovered using the RECOVER DROPPED TABLE option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for a non-nullable generated column will be rejected (SQL3116W). When this modifier is used, the table will be placed in Set Integrity Pending state. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > GENERATED COLUMN IMMEDIATE UNCHECKED`
	
	To take the table out of Set Integrity Pending state and force verification of the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > IMMEDIATE CHECKED.`
	
	When this modifier is specified and there is a generated column in any of the partitioning keys, dimension keys or distribution keys, then the LOAD command will automatically convert the modifier to `generatedignore` and proceed with the load. This will have the effect of regenerating all of the generated column values.
	
	This modifier cannot be used with either the `generatedmissing` or the `generatedignore` modifier.

identityignore	This modifier informs the load utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the `identitymissing` or the `identityoverride` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with either the `identityignore` or the `identityoverride` modifier.
identityoverride	This modifier should be used only when an identity column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of identity columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the identity column will be rejected (SQL3116W). This modifier cannot be used with either the `identitymissing` or the `identityignore` modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's identity column when this option is used.
indexfreespace=*x*	*x* is an integer between 0 and 99 inclusive. The value is interpreted as the percentage of each index page that is to be left as free space when load rebuilds the index. Load with INDEXING MODE INCREMENTAL ignores this option. The first entry in a page is added without restriction; subsequent entries are added to maintain the percent free space threshold. The default value is the one used at CREATE INDEX time. This value takes precedence over the PCTFREE value specified in the CREATE INDEX statement. The indexfreespace option affects index leaf pages only.

Table 41. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB data. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column. This option is not supported in conjunction with the CURSOR filetype. The LOBS FROM clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The LOBS FROM clause will implicitly activate the LOBSINFILE behavior. The LOBS FROM clause conveys to the LOAD utility the list of paths to search for the LOB files while loading the data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
noheader	Skips the header verification code (applicable only to load operations into tables that reside in a single-partition database partition group). If the default MPP load (mode PARTITION_AND_LOAD) is used against a table residing in a single-partition database partition group, the file is not expected to have a header. Thus the noheader modifier is not needed. If the LOAD_ONLY mode is used, the file is expected to have a header. The only circumstance in which you should need to use the noheader modifier is if you wanted to perform LOAD_ONLY operation using a file that does not have a header.
norowwarnings	Suppresses all warnings about rejected rows.
pagefreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of each data page that is to be left as free space. If the specified value is invalid because of the minimum row size, (for example, a row that is at least 3 000 bytes long, and an *x* value of 50), the row will be placed on a new page. If a value of 100 is specified, each row will reside on a new page. The PCTFREE value of a table determines the amount of free space designated per page. If a pagefreespace value on the load operation or a PCTFREE value on a table have not been set, the utility will fill up as much space as possible on each page. The value set by pagefreespace overrides the PCTFREE value specified for the table.
rowchangetimestampignore	This modifier informs the load utility that data for the row change timestamp column is present in the data file but should be ignored. This results in all ROW CHANGE TIMESTAMPs being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the `rowchangetimestampmissing` or the `rowchangetimestampoverride` modifier.
rowchangetimestampmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the row change timestamp column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT columns. This modifier cannot be used with either the `rowchangetimestampignore` or the `rowchangetimestampoverride` modifier.

Table 41. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
rowchangetimestampoverride	This modifier should be used only when a row change timestamp column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of row change timestamp columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the ROW CHANGE TIMESTAMP column will be rejected (SQL3116W). This modifier cannot be used with either the `rowchangetimestampmissing` or the `rowchangetimestampignore` modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's row change timestamp column when this option is used.

seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. LOAD converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3242W) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W) is returned.
	
	This modifier cannot be specified if the seclabelname modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might look like this:
	`"CONFIDENTIAL:ALPHA:G2"`
	`"CONFIDENTIAL;SIGMA:G2"`
	`"TOP SECRET:ALPHA:G2"`
	
	To load or import this data, the SECLABELCHAR file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELCHAR INSERT INTO t1`

seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. LOAD will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned.
	
	This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might consist of security label names similar to:
	`"LABEL1"`
	`"LABEL1"`
	`"LABEL2"`
	
	To load or import this data, the `SECLABELNAME` file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELNAME INSERT INTO t1`
	
	Note: If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.

Table 41. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
totalfreespace=*x*	*x* is an integer greater than or equal to 0 . The value is interpreted as the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if *x* is 20, and the table has 100 data pages after the data has been loaded, 20 additional empty pages will be appended. The total number of data pages for the table will be 120. The data pages total does not factor in the number of index pages in the table. This option does not affect the index object. If two loads are done with this option specified, the second load will not reuse the extra space appended to the end by the first load.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 42. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data (and numeric data specified in characters) from this code page to the database code page during the load operation. The following rules apply: • For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • For DEL data specified in an EBCDIC code page, the delimiters might not coincide with the shift-in and shift-out DBCS characters. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. EBCDIC data can use the corresponding symbols, even though the code points will be different. This option is not supported in conjunction with the CURSOR filetype.

Table 42. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

dateformat="*x*"	*x* is the format of the date in the source file.[1] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M - Month (one or two digits ranging from 1 - 12)`
	`MM - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D - Day (one or two digits ranging from 1 - 31)`
	`DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

dumpfile = *x*	*x* is the fully qualified (according to the server database partition) name of an exception file to which rejected rows are written. A maximum of 32 KB of data is written per record. Following is an example that shows how to specify a dump file: `db2 load from data of del modified by dumpfile = /u/user/filename insert into table_name` The file will be created and owned by the instance owner. To override the default file permissions, use the `dumpfileaccessall` file type modifier. **Note:** 1. In a partitioned database environment, the path should be local to the loading database partition, so that concurrently running load operations do not attempt to write to the same file. 2. The contents of the file are written to disk in an asynchronous buffered mode. In the event of a failed or an interrupted load operation, the number of records committed to disk cannot be known with certainty, and consistency cannot be guaranteed after a LOAD RESTART. The file can only be assumed to be complete for a load operation that starts and completes in a single pass. 3. If the specified file already exists, it will not be recreated, but it will be appended.
dumpfileaccessall	Grants read access to 'OTHERS' when a dump file is created. This file type modifier is only valid when: 1. it is used in conjunction with `dumpfile` file type modifier 2. the user has SELECT privilege on the load target table 3. it is issued on a DB2 server database partition that resides on a UNIX operating system If the specified file already exists, its permissions will not be changed.
fastparse	Use with caution. Reduces syntax checking on user-supplied column values, and enhances performance. Tables are guaranteed to be architecturally correct (the utility performs sufficient data checking to prevent a segmentation violation or trap), however, the coherence of the data is not validated. Only use this option if you are certain that your data is coherent and correct. For example, if the user-supplied data contains an invalid timestamp column value of `:1>0-00-20-07.11.12.000000`, this value is inserted into the table if FASTPARSE is specified, and rejected if FASTPARSE is not specified.

Table 42. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`. This modifier cannot be used with the `packeddecimal` modifier.

timeformat="*x*"	*x* is the format of the time in the source file.[1] Valid time elements are:
	`H     - Hour (one or two digits ranging from 0 - 12`
	`        for a 12 hour system, and 0 - 24`
	`        for a 24 hour system)`
	`HH    - Hour (two digits ranging from 0 - 12`
	`        for a 12 hour system, and 0 - 24`
	`        for a 24 hour system; mutually exclusive`
	`        with H)`
	`M     - Minute (one or two digits ranging`
	`        from 0 - 59)`
	`MM    - Minute (two digits ranging from 0 - 59;`
	`        mutually exclusive with M)`
	`S     - Second (one or two digits ranging`
	`        from 0 - 59)`
	`SS    - Second (two digits ranging from 0 - 59;`
	`        mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits`
	`        ranging from 00000 - 86399; mutually`
	`        exclusive with other time elements)`
	`TT    - Meridian indicator (AM or PM)`
	
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 42. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
timestampformat="*x*"	*x* is the format of the time stamp in the source file.[1] Valid time stamp elements are: YYYY - Year (four digits ranging from 0000 - 9999) M - Month (one or two digits ranging from 1 - 12) MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM) MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM) D - Day (one or two digits ranging from 1 - 31) DD - Day (two digits ranging from 1 - 31; mutually exclusive with D) DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements) H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system) HH - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H) M - Minute (one or two digits ranging from 0 - 59) MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute) S - Second (one or two digits ranging from 0 - 59) SS - Second (two digits ranging from 0 - 59; mutually exclusive with S) SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements) UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements) UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microseond elements) UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microseond elements) UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microseond elements) UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microseond elements) U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microseond elements) TT - Meridian indicator (AM or PM)

Table 42. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*" (Continued)	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.
	If the TIMESTAMPFORMAT modifier is not specified, the load utility formats the timestamp field using one of two possible formats:
	`YYYY-MM-DD-HH.MM.SS`
	`YYYY-MM-DD HH:MM:SS`
	The load utility chooses the format by looking at the separator between the DD and HH. If it is a dash '-', the load utility uses the regular dashes and dots format (YYYY-MM-DD-HH.MM.SS). If it is a blank space, then the load utility expects a colon ':' to separate the HH, MM and SS.
	In either format, if you include the microseconds field (UUUUUU), the load utility expects the dot '.' as the separator. Either YYYY-MM-DD-HH.MM.SS.UUUUUU or YYYY-MM-DD HH:MM:SS.UUUUUU are acceptable.
	The following example illustrates how to load data containing user defined date and time formats into a table called schedule:
	`db2 load from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being loaded into graphic or double-byte character large object (DBCLOB) data field(s) is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic codepage is associated with the character code page. LOAD determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the database if the `codepage` modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The `usegraphiccodepage` modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The `usegraphiccodepage` modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.

Table 42. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the `codepage` file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute.

Table 43. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII)

Modifier	Description
binarynumerics	Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions. This option is supported only with positional ASC, using fixed length records specified by the `reclen` option. The following rules apply: • No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT. • Data lengths must match their target column definitions. • FLOATs must be in IEEE Floating Point format. • Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running. NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a NULL value to *x*. The default value of *x* is `Y`.[2] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the NULL indicator character is specified to be the letter `N`, then `n` is also recognized as a NULL indicator.

Table 43. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII) (continued)

Modifier	Description

packeddecimal	Loads packed-decimal data directly, since the `binarynumerics` modifier does not include the DECIMAL field type.
	
	This option is supported only with positional ASC, using fixed length records specified by the `reclen` option.
	
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
	
	Regardless of the server platform, the byte order of binary data in the load source file is assumed to be big-endian; that is, when using this modifier on Windows operating systems, the byte order must not be reversed.
	
	This modifier cannot be used with the `implieddecimal` modifier.

reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.
striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept. This option cannot be specified together with `striptnulls`. These are mutually exclusive options. This option replaces the obsolete `t` option, which is supported for earlier compatibility only.
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. This option replaces the obsolete `padwithzero` option, which is supported for earlier compatibility only.

zoneddecimal	Loads zoned decimal data, since the BINARYNUMERICS modifier does not include the DECIMAL field type. This option is supported only with positional ASC, using fixed length records specified by the RECLEN option.
	
	Half-byte sign values can be one of the following:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	Supported values for digits are `0x0` to `0x9`.
	
	Supported values for zones are `0x3` and `0xF`.

Table 44. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII)

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark (″). The specified character is used in place of double quotation marks to enclose a character string.[23] If you want to explicitly specify the double quotation mark (″) as the character string delimiter, you should specify it as follows:
	`modified by chardel""`
	The single quotation mark (′) can also be specified as a character string delimiter as follows:
	`modified by chardel''`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[23]
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[23]

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 load ... modified by delprioritychar ...`
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
	The following example illustrates how to load data into a table called TABLE1, while preserving all leading and trailing spaces in the data file:
	`db2 load from delfile3 of del`
	`modified by keepblanks`
	`insert into table1`

nochardel	The load utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx`, `delprioritychar` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 45. Valid file type modifiers for the load utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to load each row.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Note:

1. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

   For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```

   Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
2. Character values provided for the chardel, coldel, or decpt file type modifiers must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
3. *Delimiter considerations for moving data* lists restrictions that apply to the characters that can be used as delimiter overrides.

4. The load utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the load operation fails, and an error code is returned.
5. When importing into a table containing an implicitly hidden row change timestamp column, the implicitly hidden property of the column is not honoured. Therefore, the rowchangetimestampmissing file type modifier *must be* specified in the import command if data for the column is not present in the data to be imported and there is no explicit column list present.

Table 46. LOAD behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Absent	All data in the file is assumed to be in the database code page, not the application code page, even if the CLIENT option is specified.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the database code page, even if the CLIENT option is specified. Graphic data is assumed to be in the code page of the database graphic data, even if the CLIENT option is specified. If the database code page is single-byte, then all data is assumed to be in the database code page. **Warning:** Graphic data will be corrupted when loaded into a single-byte database.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.

db2Load - Load data into a table

Loads data into a DB2 table. Data residing on the server may be in the form of a file, cursor, tape, or named pipe. Data residing on a remotely connected client may be in the form of a fully qualified file, a cursor, or named pipe. Although faster than the import utility, the load utility does not support loading data at the hierarchy level or loading into a nickname.

Authorization

One of the following:

- *sysadm*
- *dbadm*
- load authority on the database and:
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)

- INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
- INSERT privilege on the exception table, if such a table is used as part of the load operation.

Note: In general, all load processes and all DB2 server processes are owned by the instance owner. All of these processes use the identification of the instance owner to access needed files. Therefore, the instance owner must have read access to the input files, regardless of who invokes the command.

Required connection

Database. If implicit connect is enabled, a connection to the default database is established. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect gateway or loop back environment.

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

API include file

```
db2ApiDf.h
```

API and data structure syntax

```
SQL_API_RC SQL_API_FN
  db2Load (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2LoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2PartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   struct sqlu_media_list *piXmlPathList;
   struct sqllob *piLongActionString;
} db2LoadStruct;

typedef SQL_STRUCTURE db2LoadUserExit
{
   db2Char iSourceUserExitCmd;
   struct db2Char *piInputStream;
   struct db2Char *piInputFileName;
   struct db2Char *piOutputFileName;
   db2Uint16 *piEnableParallelism;
} db2LoadUserExit;
```

```
typedef SQL_STRUCTURE db2LoadIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   char *piUseTablespace;
   db2Uint32 iSavecount;
   db2Uint32 iDataBufferSize;
   db2Uint32 iSortBufferSize;
   db2Uint32 iWarningcount;
   db2Uint16 iHoldQuiesce;
   db2Uint16 iCpuParallelism;
   db2Uint16 iDiskParallelism;
   db2Uint16 iNonrecoverable;
   db2Uint16 iIndexingMode;
   db2Uint16 iAccessLevel;
   db2Uint16 iLockWithForce;
   db2Uint16 iCheckPending;
   char iRestartphase;
   char iStatsOpt;
   db2Uint16 *piXmlParse;
   db2DMUXmlValidate *piXmlValidate;
   db2Uint16 iSetIntegrityPending;
   struct db2LoadUserExit *piSourceUserExit;
} db2LoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsLoaded;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsDeleted;
   db2Uint64 oRowsCommitted;
} db2LoadOut;

typedef SQL_STRUCTURE db2PartLoadIn
{
   char *piHostname;
   char *piFileTransferCmd;
   char *piPartFileLocation;
   struct db2LoadNodeList *piOutputNodes;
   struct db2LoadNodeList *piPartitioningNodes;
   db2Uint16 *piMode;
   db2Uint16 *piMaxNumPartAgents;
   db2Uint16 *piIsolatePartErrs;
   db2Uint16 *piStatusInterval;
   struct db2LoadPortRange *piPortRange;
   db2Uint16 *piCheckTruncation;
   char *piMapFileInput;
   char *piMapFileOutput;
   db2Uint16 *piTrace;
   db2Uint16 *piNewline;
   char *piDistfile;
   db2Uint16 *piOmitHeader;
   SQL_PDB_NODE_TYPE *piRunStatDBPartNum;
} db2PartLoadIn;

typedef SQL_STRUCTURE db2LoadNodeList
{
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint16 iNumNodes;
} db2LoadNodeList;

typedef SQL_STRUCTURE db2LoadPortRange
{
   db2Uint16 iPortMin;
```

```
   db2Uint16 iPortMax;
} db2LoadPortRange;

typedef SQL_STRUCTURE db2PartLoadOut
{
   db2Uint64 oRowsRdPartAgents;
   db2Uint64 oRowsRejPartAgents;
   db2Uint64 oRowsPartitioned;
   struct db2LoadAgentInfo *poAgentInfoList;
   db2Uint32 iMaxAgentInfoEntries;
   db2Uint32 oNumAgentInfoEntries;
} db2PartLoadOut;

typedef SQL_STRUCTURE db2LoadAgentInfo
{
   db2int32 oSqlcode;
   db2Uint32 oTableState;
   SQL_PDB_NODE_TYPE oNodeNum;
   db2Uint16 oAgentType;
} db2LoadAgentInfo;

SQL_API_RC SQL_API_FN
  db2gLoad (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2gLoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2gPartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   db2Uint16 iFileTypeLen;
   db2Uint16 iLocalMsgFileLen;
   db2Uint16 iTempFilesPathLen;
   struct sqlu_media_list *piXmlPathList;
   struct sqllob *piLongActionString;
} db2gLoadStruct;

typedef SQL_STRUCTURE db2gLoadIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   char *piUseTablespace;
   db2Uint32 iSavecount;
   db2Uint32 iDataBufferSize;
   db2Uint32 iSortBufferSize;
   db2Uint32 iWarningcount;
   db2Uint16 iHoldQuiesce;
   db2Uint16 iCpuParallelism;
   db2Uint16 iDiskParallelism;
   db2Uint16 iNonrecoverable;
   db2Uint16 iIndexingMode;
   db2Uint16 iAccessLevel;
```

```
   db2Uint16 iLockWithForce;
   db2Uint16 iCheckPending;
   char iRestartphase;
   char iStatsOpt;
   db2Uint16 iUseTablespaceLen;
   db2Uint16 iSetIntegrityPending;
   db2Uint16 *piXmlParse;
   db2DMUXmlValidate *piXmlValidate;
   struct db2LoadUserExit *piSourceUserExit;
} db2gLoadIn;

typedef SQL_STRUCTURE db2gPartLoadIn
{
   char *piHostname;
   char *piFileTransferCmd;
   char *piPartFileLocation;
   struct db2LoadNodeList *piOutputNodes;
   struct db2LoadNodeList *piPartitioningNodes;
   db2Uint16 *piMode;
   db2Uint16 *piMaxNumPartAgents;
   db2Uint16 *piIsolatePartErrs;
   db2Uint16 *piStatusInterval;
   struct db2LoadPortRange *piPortRange;
   db2Uint16 *piCheckTruncation;
   char *piMapFileInput;
   char *piMapFileOutput;
   db2Uint16 *piTrace;
   db2Uint16 *piNewline;
   char *piDistfile;
   db2Uint16 *piOmitHeader;
   void *piReserved1;
   db2Uint16 iHostnameLen;
   db2Uint16 iFileTransferLen;
   db2Uint16 iPartFileLocLen;
   db2Uint16 iMapFileInputLen;
   db2Uint16 iMapFileOutputLen;
   db2Uint16 iDistfileLen;
} db2gPartLoadIn;


/* Definitions for iUsing value of db2DMUXmlValidate structure               */
#define DB2DMU_XMLVAL_XDS              1         /* Use XDS                  */
#define DB2DMU_XMLVAL_SCHEMA           2         /* Use a specified schema   */
#define DB2DMU_XMLVAL_SCHEMALOC_HINTS  3         /* Use schemaLocation hints */
#define DB2DMU_XMLVAL_ORIGSCHEMA       4         /* Use schema that document was
                                                    originally validated against
                                                    (load from cursor only) */
```

db2Load API parameters

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2LoadStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2LoadStruct data structure parameters

piSourceList
Input. A pointer to an sqlu_media_list structure used to provide a list of source files, devices, vendors, pipes, or SQL statements.

The information provided in this structure depends on the value of the media_type field. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_SQL_STMT
If the media_type field is set to this value, the caller provides an SQL query through the pStatement field of the target field. The pStatement field is of type sqlu_statement_entry. The sessions field must be set to the value of 1, since the load utility only accepts a single SQL query per load.

SQLU_SERVER_LOCATION
If the media_type field is set to this value, the caller provides information through sqlu_location_entry structures. The sessions field indicates the number of sqlu_location_entry structures provided. This is used for files, devices, and named pipes.

SQLU_CLIENT_LOCATION
If the media_type field is set to this value, the caller provides information through sqlu_location_entry structures. The sessions field indicates the number of sqlu_location_entry structures provided. This is used for fully qualified files and named pipes. Note that this media_type is only valid if the API is being called via a remotely connected client.

SQLU_TSM_MEDIA
If the media_type field is set to this value, the sqlu_vendor structure is used, where filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_OTHER_MEDIA
If the media_type field is set to this value, the sqlu_vendor structure is used, where shr_lib is the shared library name, and filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_REMOTEFETCH
If the media_type field is set to this value, the caller provides information through an sqlu_remotefetch_entry structure. The sessions field must be set to the value of 1.

piLobPathList
Input. A pointer to an sqlu_media_list structure. For IXF, ASC, and DEL file types, a list of fully qualified paths or devices to identify the location of the individual LOB files to be loaded. The file names are found in the IXF, ASC, or DEL files, and are appended to the paths provided.

The information provided in this structure depends on the value of the media_type field. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_LOCAL_MEDIA
If set to this value, the caller provides information through sqlu_media_entry structures. The sessions field indicates the number of sqlu_media_entry structures provided.

SQLU_TSM_MEDIA
If set to this value, the sqlu_vendor structure is used, where filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_OTHER_MEDIA
If set to this value, the sqlu_vendor structure is used, where shr_lib is the shared library name, and filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

piDataDescriptor
Input. Pointer to an sqldcol structure containing information about the columns being selected for loading from the external file.

If the piFileType parameter is set to SQL_ASC, the dcolmeth field of this structure must be set to SQL_METH_L. The user specifies the start and end locations for each column to be loaded.

If the file type is SQL_DEL, dcolmeth can be either SQL_METH_P or SQL_METH_D. If it is SQL_METH_P, the user must provide the source column position. If it is SQL_METH_D, the first column in the file is loaded into the first column of the table, and so on.

If the file type is SQL_IXF, dcolmeth can be one of SQL_METH_P, SQL_METH_D, or SQL_METH_N. The rules for DEL files apply here, except that SQL_METH_N indicates that file column names are to be provided in the sqldcol structure.

piActionString
Deprecated, replaced by piLongActionString.

piLongActionString
Input. Pointer to an *sqllob* structure containing a 4-byte long field, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE KEEPDICTIONARY|REPLACE RESETDICTIONARY|RESTART|TERMINATE
INTO tbname [(column_list)]
[FOR EXCEPTION e_tbname]"
```

INSERT
Adds the loaded data to the table without changing the existing table data.

REPLACE
Deletes all existing data from the table, and inserts the loaded data. The table definition and the index definitions are not changed.

RESTART
Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase.

TERMINATE
Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects may be marked as invalid, in which case index rebuild will automatically take place at next access). If the table spaces in which the table resides are not in load pending state, this option does not affect the state of the table spaces.

The load terminate option will not remove a backup pending state from table spaces.

tbname
The name of the table into which the data is to be loaded. The table cannot be a system table or a declared temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form schema.tablename. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

(column_list)
A list of table column names into which the data is to be inserted. The column names must be separated by commas. If a name contains spaces or lowercase characters, it must be enclosed by quotation marks.

FOR EXCEPTION e_tbname
Specifies the exception table into which rows in error will be copied. The exception table is used to store copies of rows that violate unique index rules, range constraints and security policies.

NORANGEEXC
Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

NOUNIQUEEXC
Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

piFileType
Input. A string that indicates the format of the input data source. Supported external formats (defined in sqlutil) are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integration Exchange Format, the preferred method for exporting data from a table so that it can be loaded later into the same table or into another database manager table.

SQL_CURSOR
An SQL query. The sqlu_media_list structure passed in through the piSourceList parameter is either of type SQLU_SQL_STMT or SQLU_REMOTEFETCH, and refers to an SQL query or a table name.

piFileTypeMod
Input. A pointer to the sqlchar structure, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link "File type modifiers for the load utility."

piLocalMsgFileName
Input. A string containing the name of a local file to which output messages are to be written.

piTempFilesPath
Input. A string containing the path name to be used on the server for temporary files. Temporary files are created to store messages, consistency points, and delete phase information.

piVendorSortWorkPaths
Input. A pointer to the sqlu_media_list structure which specifies the Vendor Sort work directories.

piCopyTargetList
Input. A pointer to an sqlu_media_list structure used (if a copy image is to be created) to provide a list of target paths, devices, or a shared library to which the copy image is to be written.

The values provided in this structure depend on the value of the media_type field. Valid values for this parameter (defined in sqlutil header file, located in the include directory) are:

SQLU_LOCAL_MEDIA
If the copy is to be written to local media, set the media_type to this value and provide information about the targets in sqlu_media_entry structures. The sessions field specifies the number of sqlu_media_entry structures provided.

SQLU_TSM_MEDIA
If the copy is to be written to TSM, use this value. No further information is required.

SQLU_OTHER_MEDIA
If a vendor product is to be used, use this value and provide further information via an sqlu_vendor structure. Set the shr_lib field of this structure to the shared library name of the vendor product. Provide only one sqlu_vendor entry, regardless of the value of sessions. The sessions field specifies the number of sqlu_media_entry structures provided. The load utility will start the sessions with different sequence numbers, but with the same data provided in the one sqlu_vendor entry.

piNullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. There is a one-to-one ordered correspondence between the elements of this array and the columns being loaded from the

data file. That is, the number of elements must equal the dcolnum field of the piDataDescriptor parameter. Each element of the array contains a number identifying a location in the data file that is to be used as a NULL indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified location in the data file must contain a Y or an N. A Y indicates that the table column data is NULL, and N indicates that the table column data is not NULL.

piLoadInfoIn
Input. A pointer to the db2LoadIn structure.

poLoadInfoOut
Output. A pointer to the db2LoadOut structure.

piPartLoadInfoIn
Input. A pointer to the db2PartLoadIn structure.

poPartLoadInfoOut
Output. A pointer to the db2PartLoadOut structure.

iCallerAction
Input. An action requested by the caller. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INITIAL
Initial call. This value (or SQLU_NOINTERRUPT) must be used on the first call to the API.

SQLU_NOINTERRUPT
Initial call. Do not suspend processing. This value (or SQLU_INITIAL) must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested load operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_ABORT
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_RESTART
Restart processing.

SQLU_DEVICE_TERMINATE
Terminate a single device. This option should be specified if the

utility is to stop reading data from the device, but further processing of the data is to be done.

piXmlPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the xml files can be found.

db2LoadUserExit data structure parameters

iSourceUserExitCmd
Input. The fully qualified name of an executable that will be used to feed data to the utility. For security reasons, the executable must be placed within the `sqllib/bin` directory on the server. This parameter is mandatory if the piSourceUserExit structure is not NULL.

The piInputStream, piInputFileName, piOutputFileName and piEnableParallelism fields are optional.

piInputStream
Input. A generic byte-stream that will be passed directly to the user-exit application via STDIN. You have complete control over what data is contained in this byte-stream and in what format. The load utility will simply carry this byte-stream over to the server and pass it into the user-exit application by feeding the process' STDIN (it will not codepage convert or modify the byte-stream). Your user-exit application would read the arguments from STDIN and use the data however intended.

One important attribute of this feature is the ability to hide sensitive information (such as userid/passwords).

piInputFileName
Input. Contains the name of a fully qualified client-side file, whose contents will be passed into the user-exit application by feeding the process' STDIN.

piOutputFileName
Input. The fully qualified name of a server-side file. The STDOUT and STDERR streams of the process which is executing the user-exit application will be streamed into this file. When piEnableParallelism is TRUE, multiple files will be created (one per user-exit instance), and each file name will be appended with a 3 digit numeric node-number value, such as `<filename>.000`).

piEnableParallelism
Input. A flag indicating that the utility should attempt to parallelize the invocation of the user-exit application.

db2LoadIn data structure parameters

iRowcount
Input. The number of physical records to be loaded. Allows a user to load only the first rowcnt rows in a file.

iRestartcount
Input. Reserved for future use.

piUseTablespace
Input. If the indexes are being rebuilt, a shadow copy of the index is built in table space iUseTablespaceName and copied over to the original table space at the end of the load. Only system temporary table spaces can be

used with this option. If not specified then the shadow index will be created in the same table space as the index object.

If the shadow copy is created in the same table space as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different table space from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load.

This field is ignored if iAccessLevel is SQLU_ALLOW_NO_ACCESS.

This option is ignored if the user does not specify INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT. This option will also be ignored if INDEXING MODE AUTOSELECT is chosen and load chooses to incrementally update the index.

iSavecount
The number of records to load before establishing a consistency point. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using db2LoadQuery - Load Query. If the value of savecount is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is 0, meaning that no consistency points will be established, unless necessary.

iDataBufferSize
The number of 4KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the required minimum is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the util_heap_sz database configuration parameter.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

iSortBufferSize
Input. This option specifies a value that overrides the SORTHEAP database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the iIndexingMode parameter is not specified as SQLU_INX_DEFERRED. The value that is specified cannot exceed the value of SORTHEAP. This parameter is useful for throttling the sort memory used by LOAD without changing the value of SORTHEAP, which would also affect general query processing.

iWarningcount
Input. Stops the load operation after warningcnt warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If warningcnt is 0, or this option is not specified, the load operation will continue regardless of the number of warnings issued.

If the load operation is stopped because the threshold of warnings was exceeded, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

iHoldQuiesce
Input. A flag whose value is set to TRUE if the utility is to leave the table in quiesced exclusive state after the load, and to FALSE if it is not.

iCpuParallelism
Input. The number of processes or threads that the load utility will create for parsing, converting and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, the load utility uses an intelligent default value at run time. Note: If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs, or the value specified by the user.

iDiskParallelism
Input. The number of processes or threads that the load utility will create for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

iNonrecoverable
Input. Set to SQLU_NON_RECOVERABLE_LOAD if the load transaction is to be marked as non-recoverable, and it will not be possible to recover it by a subsequent roll forward action. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward is completed, such a table can only be dropped. With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. Set to SQLU_RECOVERABLE_LOAD if the load transaction is to be marked as recoverable.

iIndexingMode
Input. Specifies the indexing mode. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INX_AUTOSELECT
LOAD chooses between REBUILD and INCREMENTAL indexing modes.

SQLU_INX_REBUILD
Rebuild table indexes.

SQLU_INX_INCREMENTAL
Extend existing indexes.

SQLU_INX_DEFERRED
Do not update table indexes.

iAccessLevel
Input. Specifies the access level. Valid values are:

SQLU_ALLOW_NO_ACCESS
Specifies that the load locks the table exclusively.

SQLU_ALLOW_READ_ACCESS
Specifies that the original data in the table (the non-delta portion) should still be visible to readers while the load is in progress. This option is only valid for load appends, such as a load insert, and will be ignored for load replace.

iLockWithForce
Input. A boolean flag. If set to TRUE load will force other applications as necessary to ensure that it obtains table locks immediately. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

SQLU_ALLOW_NO_ACCESS loads may force conflicting applications at the start of the load operation. At the start of the load, the utility may force applications that are attempting to either query or modify the table.

SQLU_ALLOW_READ_ACCESS loads may force conflicting applications at the start or end of the load operation. At the start of the load, the load utility may force applications that are attempting to modify the table. At the end of the load, the load utility may force applications that are attempting to either query or modify the table.

iCheckPending
This parameter is being deprecated as of Version 9.1. Use the iSetIntegrityPending parameter instead.

iRestartphase
Input. Reserved. Valid value is a single space character ' '.

iStatsOpt
Input. Granularity of statistics to collect. Valid values are:

SQLU_STATS_NONE
No statistics to be gathered.

SQLU_STATS_USE_PROFILE
Statistics are collected based on the profile defined for the current table. This profile must be created using the RUNSTATS command. If no profile exists for the current table, a warning is returned and no statistics are collected.

iSetIntegrityPending
Input. Specifies to put the table into set integrity pending state. If the value SQLU_SI_PENDING_CASCADE_IMMEDIATE is specified, set integrity pending state will be immediately cascaded to all dependent and descendent tables. If the value SQLU_SI_PENDING_CASCADE_DEFERRED is specified, the cascade of set integrity pending state to dependent tables will be deferred until the target table is checked for integrity violations. SQLU_SI_PENDING_CASCADE_DEFERRED is the default value if the option is not specified.

piSourceUserExit
Input. A pointer to the db2LoadUserExit structure.

piXmlParse
Input. Type of parsing that should occur for XML documents. Valid values found in the db2ApiDf header file in the include directory are:

DB2DMU_XMLPARSE_PRESERVE_WS
Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS
Whitespace should be stripped.

piXmlValidate
Input. Pointer to the db2DMUXmlValidate structure. Indicates that XML schema validation should occur for XML documents.

```
/* XML Validate structure                                              */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16                        iUsing;      /* What to use to perform   */
                                                 /* validation               */
   struct db2DMUXmlValidateXds    *piXdsArgs;    /* Arguments for            */
                                                 /* XMLVALIDATE USING XDS    */
   struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for            */
                                                 /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

db2LoadOut data structure parameters

oRowsRead
Output. Number of records read during the load operation.

oRowsSkipped
Output. Number of records skipped before the load operation begins.

oRowsLoaded
Output. Number of rows loaded into the target table.

oRowsRejected
Output. Number of records that could not be loaded.

oRowsDeleted
Output. Number of duplicate rows deleted.

oRowsCommitted
Output. The total number of processed records: the number of records loaded successfully and committed to the database, plus the number of skipped and rejected records.

db2PartLoadIn data structure parameters

piHostname
Input. The hostname for the iFileTransferCmd parameter. If NULL, the hostname will default to ″nohost″. This parameter is deprecated.

piFileTransferCmd
Input. File transfer command parameter. If not required, it must be set to NULL. This parameter is deprecated. Use the piSourceUserExit parameter instead.

piPartFileLocation
Input. In PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the partitioned files. This location must exist on each database partition specified by the piOutputNodes option.

For the SQL_CURSOR file type, this parameter cannot be NULL and the location does not refer to a path, but to a fully qualified file name. This will be the fully qualified base file name of the partitioned files that are created on each output database partition for PARTITION_ONLY mode, or the location of the files to be read from each database partition for

LOAD_ONLY mode. For PARTITION_ONLY mode, multiple files may be created with the specified base name if there are LOB columns in the target table. For file types other than SQL_CURSOR, if the value of this parameter is NULL, it will default to the current directory.

piOutputNodes
Input. The list of Load output database partitions. A NULL indicates that all nodes on which the target table is defined.

piPartitioningNodes
Input. The list of partitioning nodes. A NULL indicates the default.

piMode
Input. Specifies the load mode for partitioned databases. Valid values (defined in db2ApiDf header file, located in the include directory) are:

- DB2LOAD_PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

- DB2LOAD_PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. For file types other than SQL_CURSOR, the name of the output file on each database partition will have the form filename.xxx, where filename is the name of the first input file specified by piSourceList and xxx is the database partition number.For the SQL_CURSOR file type, the name of the output file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY
Data is assumed to be already distributed; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. For file types other than SQL_CURSOR, the input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number. For the SQL_CURSOR file type, the name of the input file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed, but the data file does not contain a database partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a

dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning will be written to the load message file for that database partition. The input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

piMaxNumPartAgents
Input. The maximum number of partitioning agents. A NULL value indicates the default, which is 25.

piIsolatePartErrs
Input. Indicates how the load operation will react to errors that occur on individual database partitions. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOAD_SETUP_ERRS_ONLY
In this mode, errors that occur on a database partition during setup, such as problems accessing a database partition or problems accessing a table space or table on a database partition, will cause the load operation to stop on the failing database partitions but to continue on the remaining database partitions. Errors that occur on a database partition while data is being loaded will cause the entire operation to fail and rollback to the last point of consistency on each database partition.

DB2LOAD_LOAD_ERRS_ONLY
In this mode, errors that occur on a database partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the database partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining database partitions until a failure occurs or until all the data is loaded. On the database partitions where all of the data was loaded, the data will not be visible following the load operation. Because of the errors in the other database partitions the transaction will be aborted. Data on all of the database partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the database partitions where the load operation completed and resume the load operation on database partitions that experienced an error.

Note: This mode cannot be used when iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified.

DB2LOAD_SETUP_AND_LOAD_ERRS
In this mode, database partition-level errors during setup or loading data cause processing to stop only on the affected database partitions. As with the DB2LOAD_LOAD_ERRS_ONLY mode, when database partition errors do occur while data is being

loaded, the data on all database partitions will remain invisible until a load restart operation is performed.

Note: This mode cannot 1be used when iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified.

DB2LOAD_NO_ISOLATION
Any error during the Load operation causes the transaction to be aborted. If this parameter is NULL, it will default to DB2LOAD_LOAD_ERRS_ONLY, unless iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified, in which case the default is DB2LOAD_NO_ISOLATION.

piStatusInterval
Input. Specifies the number of megabytes (MB) of data to load before generating a progress message. Valid values are whole numbers in the range of 1 to 4000. If NULL is specified, a default value of 100 will be used.

piPortRange
Input. The TCP port range for internal communication. If NULL, the port range used will be 6000-6063.

piCheckTruncation
Input. Causes Load to check for record truncation at Input/Output. Valid values are TRUE and FALSE. If NULL, the default is FALSE.

piMapFileInput
Input. Distribution map input filename. If the mode is not ANALYZE, this parameter should be set to NULL. If the mode is ANALYZE, this parameter must be specified.

piMapFileOutput
Input. Distribution map output filename. The rules for piMapFileInput apply here as well.

piTrace
Input. Specifies the number of records to trace when you need to review a dump of all the data conversion process and the output of hashing values. If NULL, the number of records defaults to 0.

piNewline
Input. Forces Load to check for newline characters at end of ASC data records if RECLEN file type modifier is also specified. Possible values are TRUE and FALSE. If NULL, the value defaults to FALSE.

piDistfile
Input. Name of the database partition distribution file. If a NULL is specified, the value defaults to "DISTFILE".

piOmitHeader
Input. Indicates that a distribution map header should not be included in the database partition file when using DB2LOAD_PARTITION_ONLY mode. Possible values are TRUE and FALSE. If NULL, the default is FALSE.

piRunStatDBPartNum
Specifies the database partition on which to collect statistics. The default value is the first database partition in the output database partition list.

db2LoadNodeList data structure parameters

piNodeList
Input. An array of node numbers.

iNumNodes
Input. The number of nodes in the piNodeList array. A 0 indicates the default, which is all nodes on which the target table is defined.

db2LoadPortRange data structure parameters

iPortMin
Input. Lower port number.

iPortMax
Input. Higher port number.

db2PartLoadOut data structure parameters

oRowsRdPartAgents
Output. Total number of rows read by all partitioning agents.

oRowsRejPartAgents
Output. Total number of rows rejected by all partitioning agents.

oRowsPartitioned
Output. Total number of rows partitioned by all partitioning agents.

poAgentInfoList
Output. During a load operation into a partitioned database, the following load processing entities may be involved: load agents, partitioning agents, pre-partitioning agents, file transfer command agents and load-to-file agents (these are described in the Data Movement Guide). The purpose of the poAgentInfoList output parameter is to return to the caller information about each load agent that participated in a load operation. Each entry in the list contains the following information:

oAgentType
A tag indicating what kind of load agent the entry describes.

oNodeNum
The number of the database partition on which the agent executed.

oSqlcode
The final sqlcode resulting from the agent's processing.

oTableState
The final status of the table on the database partition on which the agent executed (relevant only for load agents).

It is up to the caller of the API to allocate memory for this list prior to calling the API. The caller should also indicate the number of entries for which they allocated memory in the iMaxAgentInfoEntries parameter. If the caller sets poAgentInfoList to NULL or sets iMaxAgentInfoEntries to 0, then no information will be returned about the load agents.

iMaxAgentInfoEntries
Input. The maximum number of agent information entries allocated by the user for poAgentInfoList. In general, setting this parameter to 3 times the number of database partitions involved in the load operation should be sufficient.

oNumAgentInfoEntries
Output. The actual number of agent information entries produced by the load operation. This number of entries will be returned to the user in the poAgentInfoList parameter as long as iMaxAgentInfoEntries is greater than or equal to oNumAgentInfoEntries. If iMaxAgentInfoEntries is less than oNumAgentInfoEntries, then the number of entries returned in poAgentInfoList is equal to iMaxAgentInfoEntries.

db2LoadAgentInfo data structure parameters

oSqlcode
Output. The final sqlcode resulting from the agent's processing.

oTableState
Output. The purpose of this output parameter is not to report every possible state of the table after the load operation. Rather, its purpose is to report only a small subset of possible tablestates in order to give the caller a general idea of what happened to the table during load processing. This value is relevant for load agents only. The possible values are:

DB2LOADQUERY_NORMAL
Indicates that the load completed successfully on the database partition and the table was taken out of the LOAD IN PROGRESS (or LOAD PENDING) state. In this case, the table still could be in SET INTEGRITY PENDING state due to the need for further constraints processing, but this will not reported as this is normal.

DB2LOADQUERY_UNCHANGED
Indicates that the load job aborted processing due to an error but did not yet change the state of the table on the database partition from whatever state it was in prior to calling db2Load. It is not necessary to perform a load restart or terminate operation on such database partitions.

DB2LOADQUERY_LOADPENDING
Indicates that the load job aborted during processing but left the table on the database partition in the LOAD PENDING state, indicating that the load job on that database partition must be either terminated or restarted.

oNodeNum
Output. The number of the database partition on which the agent executed.

oAgentType
Output. The agent type. Valid values (defined in db2ApiDf header file, located in the include directory) are :

- DB2LOAD_LOAD_AGENT
- DB2LOAD_PARTITIONING_AGENT
- DB2LOAD_PRE_PARTITIONING_AGENT
- DB2LOAD_FILE_TRANSFER_AGENT
- DB2LOAD_LOAD_TO_FILE_AGENT

db2gLoadStruct data structure specific parameters

iFileTypeLen
Input. Specifies the length in bytes of iFileType parameter.

iLocalMsgFileLen
Input. Specifies the length in bytes of iLocalMsgFileName parameter.

iTempFilesPathLen
Input. Specifies the length in bytes of iTempFilesPath parameter.

piXmlPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the xml files can be found.

db2gLoadIn data structure specific parameters

iUseTablespaceLen
Input. The length in bytes of piUseTablespace parameter.

piXmlParse
Input. Type of parsing that should occur for XML documents. Valid values found in the db2ApiDf header file in the include directory are:

DB2DMU_XMLPARSE_PRESERVE_WS
Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS
Whitespace should be stripped.

piXmlValidate
Input. Pointer to the db2DMUXmlValidate structure. Indicates that XML schema validation should occur for XML documents.

```
/* XML Validate structure                                              */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16                       iUsing;       /* What to use to perform   */
                                                 /* validation               */
   struct db2DMUXmlValidateXds    *piXdsArgs;    /* Arguments for            */
                                                 /* XMLVALIDATE USING XDS    */
   struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for            */
                                                 /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

db2gPartLoadIn data structure specific parameters

piReserved1
Reserved for future use.

iHostnameLen
Input. The length in bytes of piHostname parameter.

iFileTransferLen
Input. The length in bytes of piFileTransferCmd parameter.

iPartFileLocLen
Input. The length in bytes of piPartFileLocation parameter.

iMapFileInputLen
Input. The length in bytes of piMapFileInput parameter.

iMapFileOutputLen
Input. The length in bytes of piMapFileOutput parameter.

iDistfileLen
Input. The length in bytes of piDistfile parameter.

Usage notes

Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted.

The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update summary tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in set integrity pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in set integrity pending state. Issue the SET INTEGRITY statement to take the tables out of set integrity pending state. Load operations cannot be carried out on replicated summary tables.

For clustering indexes, the data should be sorted on the clustering index prior to loading. The data need not be sorted when loading into an multi-dimensionally clustered (MDC) table.

Load sessions - CLP examples

Example 1

TABLE1 has 5 columns:

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:

- ELE1 positions 01 to 20
- ELE2 positions 21 to 22
- ELE3 positions 23 to 23
- ELE4 positions 24 to 27
- ELE5 positions 28 to 31
- ELE6 positions 32 to 32
- ELE7 positions 33 to 40

Data Records:

```
1...5...10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    XXN
Test data 4,5 and 6 WWN6789    Y
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc modified by striptblanks reclen=40
   method L (1 20, 21 22, 24 27, 28 31)
   null indicators (0,0,23,32)
   insert into table1 (col1, col5, col2, col3)
```

Note:

1. The specification of `striptblanks` in the MODIFIED BY parameter forces the truncation of blanks in VARCHAR columns (COL1, for example, which is 11, 17 and 19 bytes long, in rows 1, 2 and 3, respectively).
2. The specification of `reclen=40` in the MODIFIED BY parameter indicates that there is no newline character at the end of each input record, and that each record is 40 bytes long. The last 8 bytes are not use to load the table.
3. Since COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
4. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(.....)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
5. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
6. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Example 2 (Using dump files)

Table FRIENDS is defined as:

```
table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

If an attempt is made to load the following data records into this table,

```
23, 24, bobby
, 45, john
4,, mary
```

the second row is rejected because the first INT is NULL, and the column definition specifies NOT NULL. Columns which contain initial characters that are not consistent with the DEL format will generate an error, and the record will be rejected. Such records can be written to a dump file.

DEL data appearing in a column outside of character delimiters is ignored, but does generate a warning. For example:

```
22,34,"bob"
24,55,"sam" sdf
```

The utility will load ″sam″ in the third column of the table, and the characters ″sdf″ will be flagged in a warning. The record is not rejected. Another example:

```
22 3, 34,"bob"
```

The utility will load `22,34,"bob"`, and generate a warning that some data in column one following the `22` was ignored. The record is not rejected.

Example 3 (Loading a table with an identity column)

TABLE1 has 4 columns:

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)

- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

Note:

1. The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.

   ```
   db2 load from datafile1.del of del replace into table1
   ```

2. To load DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:

   ```
   db2 load from datafile1.del of del method P(1, 3, 4)
      replace into table1 (c1, c3, c4)
   db2load from datafile1.del of del modified by identityignore
      replace into table1
   ```

3. To load DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

   ```
   db2 load from datafile2.del of del replace into table1 (c1, c3, c4)
   db2 load from datafile2.del of del modified by identitymissing
      replace into table1
   ```

4. To load DATAFILE1 into TABLE2 so that the identity values of 100 and 101 are assigned to rows 3 and 4, issue the following command:

   ```
   db2 load from datafile1.del of del modified by identityoverride
      replace into table2
   ```

   In this case, rows 1 and 2 will be rejected, because the utility has been instructed to override system-generated identity values in favor of user-supplied values. If user-supplied values are not present, however, the row must be rejected, because identity columns are implicitly not NULL.

5. If DATAFILE1 is loaded into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be loaded, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

Example 4 (Loading from CURSOR)

MY.TABLE1 has 3 columns:

- ONE INT
- TWO CHAR(10)
- THREE DATE

MY.TABLE2 has 3 columns:

- ONE INT
- TWO CHAR(10)
- THREE DATE

Cursor MYCURSOR is defined as follows:

```
declare mycursor cursor for select * from my.table1
```

The following command loads all the data from MY.TABLE1 into MY.TABLE2:

```
load from mycursor of cursor method P(1,2,3) insert into
  my.table2(one,two,three)
```

Note:

1. Only one cursor name can be specified in a single LOAD command. That is, `load from mycurs1, mycurs2 of cursor...` is not allowed.
2. `P` and `N` are the only valid METHOD values for loading from a cursor.
3. In this example, METHOD P and the insert column list `(one,two,three)` could have been omitted since they represent default values.
4. MY.TABLE1 can be a table, view, alias, or nickname.

SET INTEGRITY

The SET INTEGRITY statement is used to:

- Bring one or more tables out of set integrity pending state (previously known as "check pending state") by performing required integrity processing on those tables.
- Bring one or more tables out of set integrity pending state without performing required integrity processing on those tables.
- Place one or more tables in set integrity pending state.
- Place one or more tables into full access state.
- Prune the contents of one or more staging tables.

When the statement is used to perform integrity processing for a table after it has been loaded or attached, the system can incrementally process the table by checking only the appended portion for constraints violations. If the subject table is a materialized query table or a staging table, and load, attach, or detach operations are performed on its underlying tables, the system can incrementally refresh the materialized query table or incrementally propagate to the staging table with only the delta portions of its underlying tables. However, there are some situations in which the system will not be able to perform such optimizations and will instead perform full integrity processing to ensure data integrity. Full integrity processing is done by checking the entire table for constraints violations, recomputing a materialized query table's definition, or marking a staging table as inconsistent. The latter implies that a full refresh of its associated materialized query table is required. There is also a situation in which you might want to explicitly request incremental processing by specifying the INCREMENTAL option.

The SET INTEGRITY statement is under transaction control.

Invocation

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

Authorization

The privileges required to execute the SET INTEGRITY statement depend on the purpose, as outlined below.

- Bringing tables out of set integrity pending state and performing the required integrity processing.

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

  – CONTROL privilege on:
    - The tables on which integrity processing is performed and, if exception tables are provided for one or more of those tables, INSERT privilege on the exception tables
    - All descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables that will implicitly be placed in set integrity pending state by the statement
  – LOAD authority (with conditions). The following conditions must all be met before LOAD authority can be considered as providing valid privileges:
    - The required integrity processing does not involve the following actions:
      • Refreshing a materialized query table
      • Propagating to a staging table
      • Updating a generated or identity column
    - If exception tables are provided for one or more tables, the required access is granted for the duration of the integrity processing to the tables on which integrity processing is performed, and to the associated exception tables. That is:
      • SELECT and DELETE privilege on each table on which integrity processing is performed, and
      • INSERT privilege on the exception tables
  – SYSADM or DBADM authority
- Bringing tables out of set integrity pending state without performing the required integrity processing.

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

  – CONTROL privilege on the tables that are being processed; CONTROL privilege on each descendent foreign key table, descendent immediate materialized query table, and descendent immediate staging table that will implicitly be placed in set integrity pending state by the statement
  – LOAD authority
  – SYSADM or DBADM authority
- Placing tables in set integrity pending state.

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

  – CONTROL privilege on:

- The specified tables, and
- The descendent foreign key tables that will be placed in set integrity pending state by the statement, and
- The descendent immediate materialized query tables that will be placed in set integrity pending state by the statement, and
- The descendent immediate staging tables that will be placed in set integrity pending state by the statement

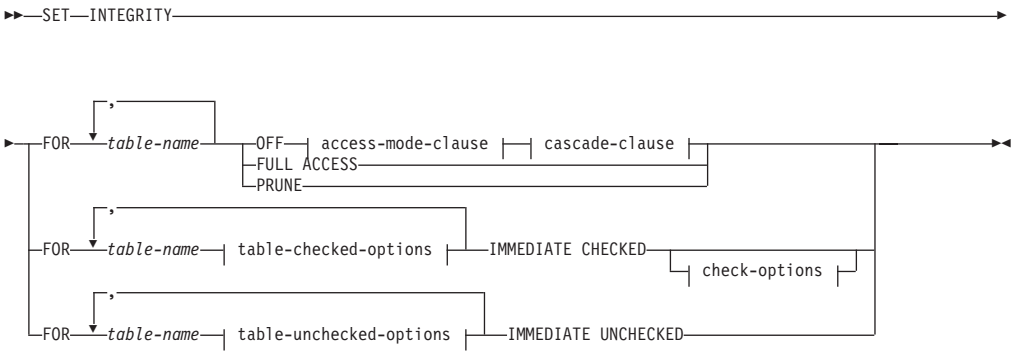
– LOAD authority
– SYSADM or DBADM authority

- Place a table into the full access state.

  The privileges held by the authorization ID of the statement must include at least one of the following:
  – CONTROL privilege on the tables that are placed into the full access state
  – LOAD authority
  – SYSADM or DBADM authority
- Prune a staging table.

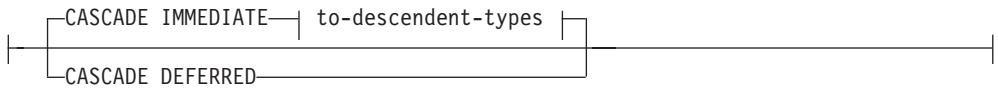
  The privileges held by the authorization ID of the statement must include at least one of the following:
  – CONTROL privilege on the table being pruned
  – SYSADM or DBADM authority

Syntax



access-mode-clause:

cascade-clause:



to-descendent-types:

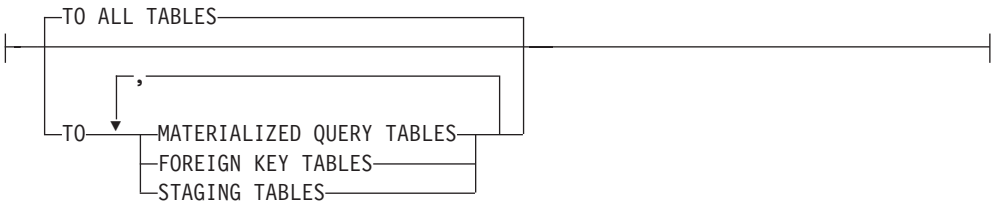
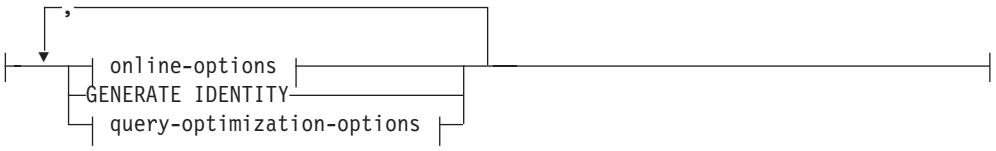


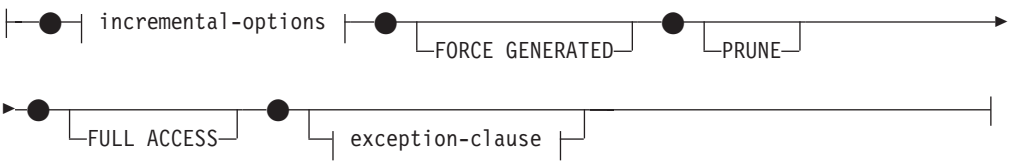
table-checked-options:



online-options:

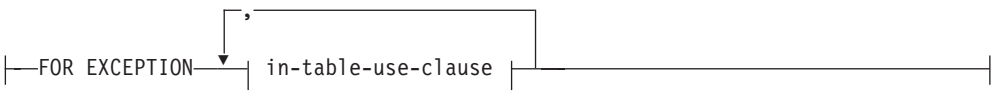
query-optimization-options:

check-options:



incremental-options:

exception-clause:



in-table-use-clause:

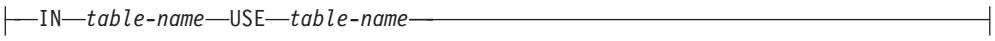
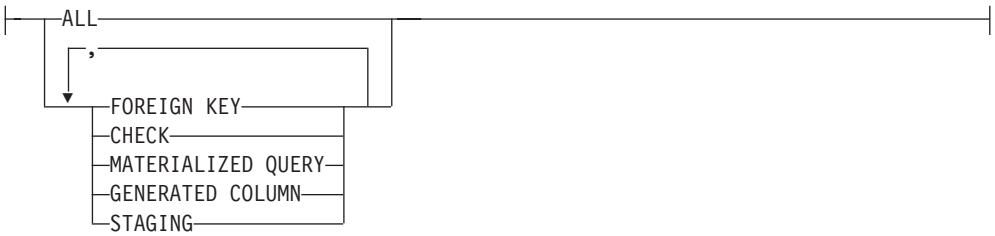


table-unchecked-options:

integrity-options:



Description

FOR *table-name*
Identifies one or more tables for integrity processing. It must be a table described in the catalog and must not be a view, catalog table, or typed table.

OFF
Specifies that the tables are placed in set integrity pending state. Only very limited activity is allowed on a table that is in set integrity pending state.

access-mode-clause
Specifies the readability of the table while it is in set integrity pending state.

NO ACCESS
Specifies that the table is to be put in set integrity pending no access state, which does not allow read or write access to the table.

READ ACCESS
Specifies that the table is to be put in set integrity pending read access state, which allows read access to the non-appended portion of the table. This option is not allowed on a table that is in set integrity pending no access state (SQLSTATE 428FH).

cascade-clause
Specifies whether the set integrity pending state of the table referenced in the SET INTEGRITY statement is to be immediately cascaded to descendent tables.

CASCADE IMMEDIATE
Specifies that the set integrity pending state is to be immediately extended to descendent tables.

to-descendent-types
Specifies the type of descendent tables to which the set integrity pending state is immediately cascaded.

TO ALL TABLES
Specifies that the set integrity pending state is to be immediately cascaded to all descendent tables of the tables in the invocation list. Descendent tables include all descendent foreign key tables, immediate

staging tables, and immediate materialized query tables that are descendants of the tables in the invocation list, or descendants of descendent foreign key tables.

Specifying TO ALL TABLES is equivalent to specifying TO FOREIGN KEY TABLES, TO MATERIALIZED QUERY TABLES, and TO STAGING TABLES, all in the same statement.

TO MATERIALIZED QUERY TABLES
If only TO MATERIALIZED QUERY TABLES is specified, the set integrity pending state is to be immediately cascaded only to descendent immediate materialized query tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO MATERIALIZED QUERY TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate materialized query tables of the tables in the invocation list, and to all immediate materialized query tables that are descendants of the descendent foreign key tables.

TO FOREIGN KEY TABLES
Specifies that the set integrity pending state is to be immediately cascaded to descendent foreign key tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state.

TO STAGING TABLES
Specifies that the set integrity pending state is to be immediately cascaded to descendent staging tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO STAGING TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate staging tables of the tables in the invocation list, and to all immediate staging tables that are descendants of the descendent foreign key tables.

CASCADE DEFERRED
Specifies that only the tables in the invocation list are to be put in set integrity pending state. The states of the descendent tables will remain unchanged. Descendent foreign key tables might later be implicitly put in set integrity pending state when their parent tables are checked for constraints violations. Descendent immediate materialized query tables and descendent immediate staging tables might be implicitly put in set integrity pending state when one of their underlying tables is checked for integrity violations.

If *cascade-clause* is not specified, the set integrity pending state is immediately cascaded to all descendent tables.

IMMEDIATE CHECKED
Specifies that the table is to be taken out of set integrity pending state by performing required integrity processing on the table. This is done in accordance with the information set in the STATUS and CONST_CHECKED columns of the SYSCAT.TABLES catalog view. That is:

- The value in the STATUS column must be 'C' (the table is in set integrity pending state), or an error is returned (SQLSTATE 51027), unless the table is a descendent foreign key table, descendent materialized query table, or

descendent staging table of a table that is specified in the list, is in set integrity pending state, and whose intermediate ancestors are also in the list.

- If the table being checked is in set integrity pending state, the value in CONST_CHECKED indicates which integrity options are to be checked.

When the table is taken out of set integrity pending state, its descendent tables are, if necessary, put in set integrity pending state. A warning to indicate that descendent tables have been put in set integrity pending state is returned (SQLSTATE 01586).

If the table is a system-maintained materialized query table, the data is checked against the query and refreshed as necessary. (IMMEDIATE CHECKED cannot be used for user-maintained materialized query tables.) If the table is a staging table, the data is checked against its query definition and propagated as necessary.

When the integrity of a child table is checked:

- None of its parents can be in set integrity pending state, or
- Each of its parents must be checked for constraints violations in the same SET INTEGRITY statement

When an immediate materialized query table is refreshed, or deltas are propagated to a staging table:

- None of its underlying tables can be in set integrity pending state, or
- Each of its underlying tables must be checked in the same SET INTEGRITY statement

Otherwise, an error is returned (SQLSTATE 428A8).

table-checked-options

online-options
Specifies the accessibility of the table while it is being processed.

ALLOW NO ACCESS
Specifies that no other users can access the table while it is being processed, except if they are using the Uncommitted Read isolation level.

ALLOW READ ACCESS
Specifies that other users have read-only access to the table while it is being processed.

ALLOW WRITE ACCESS
Specifies that other users have read and write access to the table while it is being processed.

GENERATE IDENTITY
Specifies that if the table includes an identity column, the values are generated by the SET INTEGRITY statement. By default, when the GENERATE IDENTITY option is specified, only attached rows will have their identity column values generated by the SET INTEGRITY statement. The NOT INCREMENTAL option must be specified in conjunction with the GENERATE IDENTITY option to have the SET INTEGRITY statement generate identity column values for all rows in the table, including attached rows, loaded rows, and existing rows. If the GENERATE IDENTITY option is not specified, the current identity column values for all rows in the table are left unchanged.

query-optimization-options
Specifies the query optimization options for the maintenance of REFRESH DEFERRED materialized query tables.

ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY
Specifies that when the CURRENT REFRESH AGE special register is set to 'ANY', the maintenance of *table-name* will allow REFRESH DEFERRED materialized query tables to be used to optimize the query that maintains *table-name*. If *table-name* is not a REFRESH DEFERRED materialized query table, an error is returned (SQLSTATE 428FH). REFRESH IMMEDIATE materialized query tables are always considered during query optimization.

check-options

incremental-options

INCREMENTAL
Specifies the application of integrity processing on the appended portion (if any) of the table. If such a request cannot be satisfied (that is, the system detects that the whole table needs to be checked for data integrity), an error is returned (SQLSTATE 55019).

NOT INCREMENTAL
Specifies the application of integrity processing on the whole table. If the table is a materialized query table, the materialized query table definition is recomputed. If the table has at least one constraint defined on it, this option causes full processing of descendent foreign key tables and descendent immediate materialized query tables. If the table is a staging table, it is set to an inconsistent state.

If the *incremental-options* clause is not specified, the system determines whether incremental processing is possible; if not, the whole table is checked.

FORCE GENERATED
If the table includes generated by expression columns, the values are computed on the basis of the expression and stored in the column. If this option is not specified, the current values are compared to the computed value of the expression, as though an equality check constraint were in effect. If the table is processed for integrity incrementally, generated columns are computed only for the appended portion.

PRUNE
This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH). If the INCREMENTAL check option is also specified, an error is returned (SQLSTATE 428FH).

FULL ACCESS
Specifies that the table is to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table (that has dependent immediate materialized query tables or dependent immediate staging tables) in the invocation

list is incrementally processed, the underlying table is put in no data movement state, as required, after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables are taken out of set integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE CHECKED option, the underlying table is put directly in full access state (bypassing the no data movement state). Dependent immediate materialized query tables that have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them might be flagged as inconsistent.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is put directly into full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option was specified.

exception-clause

FOR EXCEPTION
Specifies that any row that is in violation of a constraint being checked is to be moved to an exception table. Even if errors are detected, the table is taken out of set integrity pending state. A warning to indicate that one or more rows have been moved to the exception tables is returned (SQLSTATE 01603).

If the FOR EXCEPTION option is not specified and any constraints are violated, only the first detected violation is returned (SQLSTATE 23514). If there is a violation in any table, all of the tables are left in set integrity pending state.

It is recommended to always use the FOR EXCEPTION option when checking for constraints violations to prevent a rollback of the SET INTEGRITY statement if a violation is found.

IN *table-name*
Specifies the table from which rows that violate constraints are to be moved. There must be one exception table specified for each table being checked. This clause cannot be specified for a materialized query table or a staging table (SQLSTATE 428A7).

USE *table-name*
Specifies the exception table into which error rows are to be moved.

FULL ACCESS
If the FULL ACCESS option is specified as the only operation of the statement, the table is placed into the full access state without being rechecked for integrity violations. However, dependent immediate materialized query tables that have not been refreshed might require a full recomputation in subsequent REFRESH TABLE statements, and dependent immediate staging tables that have not had the delta portions of the table propagated to them might be changed to incomplete state. This option can only be specified for a table that is in the no data movement state or the no access state, but not in the set integrity pending state (SQLSTATE 428FH).

PRUNE
This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH).

table-unchecked-options

integrity-options
Used to define the types of required integrity processing that are to be bypassed when the table is taken out of the set integrity pending state.

ALL
The table will be immediately taken out of set integrity pending state without any of its required integrity processing being performed.

FOREIGN KEY
Required foreign key constraints checking will not be performed when the table is brought out of set integrity pending state.

CHECK
Required check constraints checking will not be performed when the table is brought out of set integrity pending state.

MATERIALIZED QUERY
Required refreshing of a materialized query table will not be performed when the table is brought out of set integrity pending state.

GENERATED COLUMN
Required generated column constraints checking will not be performed when the table is brought out of set integrity pending state.

STAGING
Required propagation of data to a staging table will not be performed when the table is brought out of set integrity pending state.

If no other types of integrity processing are required on the table after a specific type of integrity processing has been marked as bypassed, the table is immediately taken out of set integrity pending state.

FULL ACCESS
Specifies that the tables are to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table in the invocation list is incrementally processed, and it has dependent immediate materialized query tables or dependent immediate staging tables, the underlying table is placed, as required, in the no data movement state after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables have been taken out of set integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE UNCHECKED option, the underlying table is placed directly in full access state (it bypasses the no data movement state). Dependent immediate materialized query tables that have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them mIGHT be flagged as inconsistent.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is placed directly in full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option has been specified.

If the FULL ACCESS option has been specified with the IMMEDIATE UNCHECKED option, and the statement does not bring the table out of set integrity pending state, an error is returned (SQLSTATE 428FH).

IMMEDIATE UNCHECKED
Specifies one of the following:

- The table is to be brought out of set integrity pending state immediately without any required integrity processing.
- The table is to have one or more types of required integrity processing bypassed when the table is brought out of set integrity pending state by a subsequent SET INTEGRITY statement using the IMMEDIATE CHECKED option.

Consider the data integrity implications of this option before using it. See the "Notes" section below.

Notes

- Effects on tables in one of the restricted set integrity-related states:
  - Use of INSERT, UPDATE, or DELETE is disallowed on a table that is in read access state or in no access state. Furthermore, any statement that requires this type of modification to a table that is in such a state will be rejected. For example, deletion of a row in a parent table that cascades to a dependent table that is in the no access state is not allowed.
  - Use of SELECT is disallowed on a table that is in the no access state. Furthermore, any statement that requires read access to a table that is in the no access state will be rejected.
  - New constraints added to a table are normally enforced immediately. However, if the table is in set integrity pending state, the checking of any new constraints is deferred until the table is taken out of set integrity pending state. If the table is in set integrity pending state, addition of a new constraint places the table into set integrity pending no access state, because validity of data is at risk.
  - The CREATE INDEX statement cannot reference any table that is in read access state or in no access state. Similarly, an ALTER TABLE statement to add a primary key or a unique constraint cannot reference any table that is in read access state or in no access state.
  - The import utility is not allowed to operate on a table that is in read access state or in no access state.
  - The export utility is not allowed to operate on a table that is in no access state, but is allowed to operate on a table that is in read access state. If a table is in read access state, the export utility will only export the data that is in the non-appended portion.
  - Operations (like REORG, REDISTRIBUTE, update distribution key, update multidimensional clustering key, update range clustering key, update table partitioning key, and so on) that might involve data movement within a table are not allowed on a table that is in any of the following states: read access, no access, or no data movement.

– The load, backup, restore, update statistics, runstats, reorgchk, list history, and rollforward utilities are allowed on a table that is in any of the following states: full access, read access, no access, or no data movement.
– The ALTER TABLE, COMMENT, DROP TABLE, CREATE ALIAS, CREATE TRIGGER, CREATE VIEW, GRANT, REVOKE, and SET INTEGRITY statements can reference a table that is in any of the following states: full access, read access, no access, or no data movement. However, they might cause the table to be put into no access state.
– Packages, views, and any other objects that depend on a table that is in no access state will return an error when the table is accessed at run time. Packages that depend on a table that is in read access state will return an error when an insert, update, or delete operation is attempted on the table at run time.

The removal of violating rows by the SET INTEGRITY statement is not a delete event. Therefore, triggers are never activated by a SET INTEGRITY statement. Similarly, updating generated columns using the FORCE GENERATED option does not activate triggers.

- Incremental processing will be used whenever the situation allows it, because it is more efficient. The INCREMENTAL option is not needed in most cases. It is needed, however, to ensure that integrity checks are indeed processed incrementally. If the system detects that full processing is needed to ensure data integrity, an error is returned (SQLSTATE 55019).
- Warning about the use of the IMMEDIATE UNCHECKED clause:
  – This clause is intended to be used by utility programs, and its use by application programs is not recommended. If there is data in the table that does not meet the integrity specifications that were defined for the table, and the IMMEDIATE UNCHECKED option is used, incorrect query results might be returned.

    The fact that the table was taken out of the set integrity pending state without performing the required integrity processing will be recorded in the catalog (the respective byte in the CONST_CHECKED column in the SYSCAT.TABLES view will be set to 'U'). This indicates that the user has assumed responsibility for data integrity with respect to the specific constraints. This value remains unchanged until either:

    - The table is put back into set integrity pending state (by referencing the table in a SET INTEGRITY statement with the OFF option), at which time 'U' values in the CONST_CHECKED column are changed to 'W' values, indicating that the user had previously assumed responsibility for data integrity, and the system needs to verify the data.
    - All unchecked constraints for the table are dropped.

    The 'W' state differs from the 'N' state in that it records the fact that integrity was previously checked by the user, but not yet by the system. If the user issues the SET INTEGRITY ... IMMEDIATE CHECKED statement with the NOT INCREMENTAL option, the system rechecks the whole table for data integrity (or performs a full refresh on a materialized query table), and then changes the 'W' state to the 'Y' state. If IMMEDIATE UNCHECKED is specified, or if NOT INCREMENTAL is not specified, the 'W' state is changed back to the 'U' state to record the fact that some data has still not been verified by the system. In the latter case (when the NOT INCREMENTAL is not specified), a warning is returned (SQLSTATE 01636).

If an underlying table's integrity has been checked using the IMMEDIATE UNCHECKED clause, the 'U' values in the CONST_CHECKED column of the underlying table will be propagated to the corresponding CONST_CHECKED column of:

- Dependent immediate materialized query tables
- Dependent deferred materialized query tables
- Dependent staging tables

For a dependent immediate materialized query table, this propagation is done whenever the underlying table is brought out of set integrity pending state, and whenever the materialized query table is refreshed. For a dependent deferred materialized query table, this propagation is done whenever the materialized query table is refreshed. For dependent staging tables, this propagation is done whenever the underlying table is brought out of set integrity pending state. These propagated 'U' values in the CONST_CHECKED columns of dependent materialized query tables and staging tables record the fact that these materialized query tables and staging tables depend on some underlying table whose required integrity processing has been bypassed using the IMMEDIATE UNCHECKED option.

For a materialized query table, the 'U' value in the CONST_CHECKED column that was propagated by the underlying table will remain until the materialized query table is fully refreshed and none of its underlying tables have a 'U' value in their corresponding CONST_CHECKED column. After such a refresh, the 'U' value in the CONST_CHECKED column for the materialized query table will be changed to 'Y'.

For a staging table, the 'U' value in the CONST_CHECKED column that was propagated by the underlying table will remain until the corresponding deferred materialized query table of the staging table is refreshed. After such a refresh, the 'U' value in the CONST_CHECKED column for the staging table will be changed to 'Y'.

– If a child table and its parent table are checked in the same SET INTEGRITY statement with the IMMEDIATE CHECKED option, and the parent table requires full checking of its constraints, the child table will have its foreign key constraints checked, independently of whether or not the child table has a 'U' value in the CONST_CHECKED column for foreign key constraints.

- After appending data using LOAD INSERT or ALTER TABLE ATTACH, the SET INTEGRITY statement with the IMMEDIATE CHECKED option checks the table for constraints violations. The system determines whether incremental processing on the table is possible. If so, only the appended portion is checked for integrity violations. If not, the system checks the whole table for integrity violations.
- Consider the statement:

```
SET INTEGRITY FOR T IMMEDIATE CHECKED
```

Situations in which the system will require a full refresh, or will check the whole table for integrity (the INCREMENTAL option cannot be specified) are:

– When new constraints have been added to T itself while it is in the set integrity pending state

– When a LOAD REPLACE operation against T, it parents, or its underlying tables has taken place

– When the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated after the last integrity check on T, its parents, or its underlying tables

  - The cascading effect of full processing, when any parent of T (or underlying table, if T is a materialized query table or a staging table) has been checked for integrity non-incrementally
  - If the table space containing the table or its parent (or underlying table of a materialized query table or a staging table) has been rolled forward to a point in time, and the table and its parent (or underlying table if the table is a materialized query table or a staging table) reside in different table spaces
  - When T is a materialized query table, and a LOAD REPLACE or LOAD INSERT operation directly into T has taken place after the last refresh
- If the conditions for full processing described in the previous bullet are not satisfied, the system will attempt to check only the appended portion for integrity, or perform an incremental refresh (if it is a materialized query table) when the user does not specify the NOT INCREMENTAL option for the statement `SET INTEGRITY FOR T IMMEDIATE CHECKED`.
- If an error occurs during integrity processing, all the effects of the processing (including deleting from the original and inserting into the exception tables) will be rolled back.
- If a SET INTEGRITY statement issued with the FORCE GENERATED option fails because of a lack of log space, increase available active log space and reissue the SET INTEGRITY statement. Alternatively, use the SET INTEGRITY statement with the GENERATED COLUMN and IMMEDIATE UNCHECKED options to bypass generated column checking for the table. Then, issue a SET INTEGRITY statement with the IMMEDIATE CHECKED option and without the FORCE GENERATED option to check the table for other integrity violations (if applicable) and to bring it out of set integrity pending state. After the table is out of the set integrity pending state, the generated columns can be updated to their default (generated) values by assigning them to the keyword DEFAULT in an UPDATE statement. This is accomplished by using either multiple searched update statements based on ranges (each followed by a commit), or a cursor-based approach using intermittent commits. A "with hold" cursor should be used if locks are to be retained after intermittent commits using the cursor-based approach.
- A table that was put into set integrity pending state using the CASCADE DEFERRED option of the SET INTEGRITY statement or the LOAD command, or through the ALTER TABLE statement with the ATTACH clause, and that is checked for integrity violations using the IMMEDIATE CHECKED option of the SET INTEGRITY statement, will have its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:
  - If the entire table is checked for integrity violations, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  - If the table is checked for integrity violations incrementally, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  - If the table requires no checking at all, its descendent immediate materialized query tables, descendent staging tables, and descendent foreign key tables will remain in their original states.
- A table that was put in set integrity pending state using the CASCADE DEFERRED option (of the SET INTEGRITY statement or the LOAD command), and that is brought out of set integrity pending state using the IMMEDIATE UNCHECKED option of the SET INTEGRITY statement, will have its descendent

foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:

  – If the table has been loaded using the REPLACE mode, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  – If the table has been loaded using the INSERT mode, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  – If the table has not been loaded, its descendent immediate materialized query tables, descendent staging tables, and its descendent foreign key tables will remain in their original states.

- SET INTEGRITY is usually a long running statement. In light of this, to reduce the risk of a rollback of the entire statement because of a lock timeout, you can issue the SET CURRENT LOCK TIMEOUT statement with the WAIT option before executing the SET INTEGRITY statement, and then reset the special register to its previous value after the transaction commits. Note, however, that the CURRENT LOCK TIMEOUT special register only impacts a specific set of lock types.
- If you use the ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY option, ensure that the maintenance order is correct for REFRESH DEFERRED materialized query tables. For example, consider two materialized query tables, MQT1 and MQT2, whose materialized queries share the same underlying tables. The materialized query for MQT2 can be calculated using MQT1, instead of the underlying tables. If separate statements are used to maintain these two materialized query tables, and MQT2 is maintained first, the system might choose to use the contents of MQT1, which has not yet been maintained, to maintain MQT2. In this case, MQT1 would contain current data, but MQT2 could still contain stale data, even though both were maintained at almost the same time. The correct maintenance order, if two SET INTEGRITY statements are used instead of one, is to maintain MQT1 first.
- When using the SET INTEGRITY statement to perform integrity processing on a base table that has been loaded or attached, it is recommended that you process its dependent REFRESH IMMEDIATE materialized query tables and its PROPAGATE IMMEDIATE staging tables in the same SET INTEGRITY statement to avoid putting these dependent tables in set integrity pending no access state at the end of SET INTEGRITY processing. Note that for base tables that have a large number of dependent REFRESH IMMEDIATE materialized query tables and PROPAGATE IMMEDIATE staging tables, memory constraints might make it impossible to process all of the dependents in the same statement as the base table.
- If the FORCE GENERATED or the GENERATE IDENTITY option is specified, and the column that is generated is part of a unique index, the SET INTEGRITY statement returns an error (SQLSTATE 23505) and rolls back if it detects duplicate keys in the unique index. This error is returned even if there is an exception table for the table being processed.

  This scenario can occur under the following circumstances:

  – The SET INTEGRITY statement runs after a LOAD command against the table, and the GENERATEDOVERRIDE or the IDENTITYOVERRIDE file type modifier is specified during the load operation. To prevent this scenario, it is recommended that you use the GENERATEDIGNORE or the GENERATEDMISSING file type modifer instead of GENERATEDOVERRIDE, and that you use the IDENTITYIGNORE or the IDENTITYMISSING modifier

instead of IDENTITYOVERRIDE. Using the recommended modifiers will prevent the need for any generated by expression column or identity column processing during SET INTEGRITY statement execution.

– The SET INTEGRITY statement is run after an ALTER TABLE statement that alters the expression of a generated by expression column.

To bring a table out of the set integrity pending state after encountering such a scenario:

– Do not use the FORCE GENERATED or the GENERATE IDENTITY option to regenerate the column values. Instead, use the IMMEDIATE CHECKED option in conjunction with the FOR EXCEPTION option to move any rows that violate the generated column expression to an exception table. Then, re-insert the rows into the table from the exception table, which will generate the correct expression and perform unique key checking. This prevents having to reprocess the entire table, because only those rows that violated the generated column expression will need to be processed again.

– If the table being processed has attached partitions, detach those partitions before performing the actions that are described in the previous bullet. Then, re-attach the partitions and execute a SET INTEGRITY statement to process integrity on the attached partitions seperately.

- If a protected table is specified for the SET INTEGRITY statement along with an exception table, all of the following table criteria must be met; otherwise, an error is returned (SQLSTATE 428A5):
  – The tables must be protected by the same security policy.
  – If a column in the protected table has data type DB2SECURITYLABEL, the corresponding column in the exception table must also have data type DB2SECURITYLABEL.
  – If a column in the protected table is protected by a security label, the corresponding column in the exception table must also be protected by the same security label.
- ***Compatibilities***
  – For compatibility with previous versions of DB2:
    - SET CONSTRAINTS can be specified in place of SET INTEGRITY
    - SUMMARY can be specified in place of MATERIALIZED QUERY

Examples

Example 1: The following is an example of a query that provides information about the set integrity pending state and the set integrity-related access restriction states of tables. SUBSTR is used to extract individual bytes of the CONST_CHECKED column of SYSCAT.TABLES. The first byte represents foreign key constraints; the second byte represents check constraints; the fifth byte represents materialized query table integrity; the sixth byte represents generated column constraints; the seventh byte represents staging table integrity; and the eighth byte represents data partitioning constraints. STATUS gives the set integrity pending state, and ACCESS_MODE gives the set integrity-related access restriction state.

```
SELECT TABNAME, STATUS, ACCESS_MODE,
  SUBSTR(CONST_CHECKED,1,1) AS FK_CHECKED,
  SUBSTR(CONST_CHECKED,2,1) AS CC_CHECKED,
  SUBSTR(CONST_CHECKED,5,1) AS MQT_CHECKED,
  SUBSTR(CONST_CHECKED,6,1) AS GC_CHECKED,
  SUBSTR(CONST_CHECKED,7,1) AS STG_CHECKED,
  SUBSTR(CONST_CHECKED,8,1) AS DP_CHECKED
FROM SYSCAT.TABLES
```

Example 2: Put the PARENT table in set integrity pending no access state, and immediately cascade the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
  NO ACCESS CASCADE IMMEDIATE
```

Example 3: Put the PARENT table in set integrity pending read access state without immediately cascading the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
  READ ACCESS CASCADE DEFERRED
```

Example 4: Check integrity for a table named FACT_TABLE. If there are no integrity violations detected, the table is brought out of set integrity pending state. If any integrity violations are detected, the entire statement is rolled back, and the table remains in set integrity pending state.

```
SET INTEGRITY FOR FACT_TABLE IMMEDIATE CHECKED
```

Example 5: Check integrity for the SALES and PRODUCTS tables, and move the rows that violate integrity into exception tables named SALES_EXCEPTIONS and PRODUCTS_EXCEPTIONS. Both the SALES and PRODUCTS tables are brought out of set integrity pending state, whether or not there are any integrity violations.

```
SET INTEGRITY FOR SALES, PRODUCTS IMMEDIATE CHECKED
  FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS,
  IN PRODUCTS USE PRODUCTS_EXCEPTIONS
```

Example 6: Enable FOREIGN KEY constraint checking in the MANAGER table, and CHECK constraint checking in the EMPLOYEE table, to be bypassed with the IMMEDIATE UNCHECKED option.

```
SET INTEGRITY FOR MANAGER FOREIGN KEY,
  EMPLOYEE CHECK IMMEDIATE UNCHECKED
```

Example 7: Add a check constraint and a foreign key to the EMP_ACT table, using two ALTER TABLE statements. The SET INTEGRITY statement with the OFF option is used to put the table in set integrity pending state, so that the constraints are not checked immediately upon execution of the two ALTER TABLE statements. The single SET INTEGRITY statement with the IMMEDIATE CHECKED option is used to check both of the added constraints during a single pass through the table.

```
SET INTEGRITY FOR EMP_ACT OFF;
ALTER TABLE EMP_ACT ADD CHECK
  (EMSTDATE <= EMENDATE);
ALTER TABLE EMP_ACT ADD FOREIGN KEY
  (EMPNO) REFERENCES EMPLOYEE;
SET INTEGRITY FOR EMP_ACT IMMEDIATE CHECKED
  FOR EXCEPTION IN EMP_ACT USE EMP_ACT_EXCEPTIONS
```

Example 8: Update generated columns with the correct values.

```
SET INTEGRITY FOR SALES IMMEDIATE CHECKED
  FORCE GENERATED
```

Example 9: Append (using LOAD INSERT) from different sources into an underlying table (SALES) of a REFRESH IMMEDIATE materialized query table (SALES_SUMMARY). Check SALES incrementally for data integrity, and refresh SALES_SUMMARY incrementally. In this scenario, integrity checking for SALES and refreshing of SALES_SUMMARY are incremental, because the system chooses incremental processing. The ALLOW READ ACCESS option is used on the SALES table to allow concurrent reads of existing data while integrity checking of the loaded portion of the table is taking place.

```
LOAD FROM 2000_DATA.DEL OF DEL
  INSERT INTO SALES ALLOW READ ACCESS;
LOAD FROM 2001_DATA.DEL OF DEL
  INSERT INTO SALES ALLOW READ ACCESS;
SET INTEGRITY FOR SALES ALLOW READ ACCESS IMMEDIATE CHECKED
  FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS;
REFRESH TABLE SALES_SUMMARY;
```

Example 10: Attach a new partition to a data partitioned table named SALES. Incrementally check for constraints violations in the attached data of the SALES table and incrementally refresh the dependent SALES_SUMMARY table. The ALLOW WRITE ACCESS option is used on both tables to allow concurrent updates while integrity checking is taking place.

```
ALTER TABLE SALES
  ATTACH PARTITION STARTING (100) ENDING (200)
  FROM SOURCE;
SET INTEGRITY FOR SALES ALLOW WRITE ACCESS, SALES_SUMMARY ALLOW WRITE ACCESS
  IMMEDIATE CHECKED FOR EXCEPTION IN SALES
  USE SALES_EXCEPTIONS;
```

Example 11: Detach a partition from a data partitioned table named SALES. Incrementally refresh the dependent SALES_SUMMARY table.

```
ALTER TABLE SALES
  DETACH PARTITION 2000_PART INTO ARCHIVE_TABLE;
SET INTEGRITY FOR SALES_SUMMARY
  IMMEDIATE CHECKED;
```

Example 12: Bring a new user-maintained materialized query table out of set integrity pending state.

```
CREATE TABLE YEARLY_SALES
  AS (SELECT YEAR, SUM(SALES)AS SALES
  FROM FACT_TABLE GROUP BY YEAR)
  DATA INITIALLY DEFERRED REFRESH DEFERRED MAINTAINED BY USER

SET INTEGRITY FOR YEARLY_SALES
  ALL IMMEDIATE UNCHECKED
```

LOAD QUERY

Checks the status of a load operation during processing and returns the table state. If a load is not processing, then the table state alone is returned. A connection to the same database, and a separate CLP session are also required to successfully invoke this command. It can be used either by local or remote users.

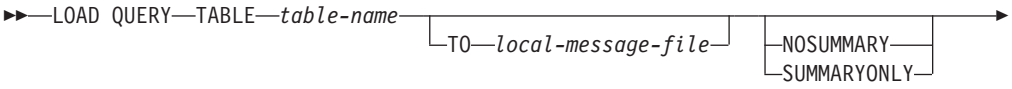
Authorization

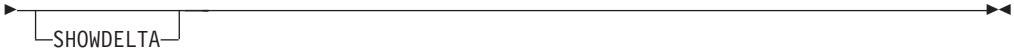
None

Required connection

Database

Command syntax





Command parameters

NOSUMMARY
Specifies that no load summary information (rows read, rows skipped, rows loaded, rows rejected, rows deleted, rows committed, and number of warnings) is to be reported.

SHOWDELTA
Specifies that only new information (pertaining to load events that have occurred since the last invocation of the LOAD QUERY command) is to be reported.

SUMMARYONLY
Specifies that only load summary information is to be reported.

TABLE *table-name*
Specifies the name of the table into which data is currently being loaded. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

TO *local-message-file*
Specifies the destination for warning and error messages that occur during the load operation. This file cannot be the *message-file* specified for the LOAD command. If the file already exists, all messages that the load utility has generated are appended to it.

Examples

A user loading a large amount of data into the STAFF table in the BILLYBOB database, wants to check the status of the load operation. The user can specify:

```
db2 connect to billybob
db2 load query table staff to /u/mydir/staff.tempmsg
```

The output file `/u/mydir/staff.tempmsg` might look like the following:

```
SQL3501W  The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.

SQL3109N  The utility is beginning to load data from file
"/u/mydir/data/staffbig.del"

SQL3500W  The utility is beginning the "LOAD" phase at time "03-21-2002
11:31:16.597045".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count = "104416".

SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count = "205757".

SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count = "307098".

SQL3520W  Load Consistency Point was successful.
```

```
SQL3519W  Begin Load Consistency Point. Input record count = "408439".

SQL3520W  Load Consistency Point was successful.

SQL3532I  The Load utility is currently in the "LOAD" phase.


Number of rows read         = 453376
Number of rows skipped      = 0
Number of rows loaded       = 453376
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 408439
Number of warnings          = 0

Tablestate:
  Load in Progress
```

Usage notes

In addition to locks, the load utility uses table states to control access to the table. The LOAD QUERY command can be used to determine the table state; LOAD QUERY can be used on tables that are not currently being loaded. For a partitioned table, the state reported is the most restrictive of the corresponding visible data partition states. For example, if a single data partition is in the Read Access Only state and all other data partitions are in Normal state, the load query operation returns the Read Access Only state. A load operation will not leave a subset of data partitions in a state different from the rest of the table. The table states described by LOAD QUERY are as follows:

Normal
A table is in Normal state if it is not in any of the other (abnormal) table states. Normal state is the initial state of a table after it is created.

Set Integrity Pending
The table has constraints which have not yet been verified. Use the SET INTEGRITY statement to take the table out of Set Integrity Pending state. The load utility places a table in Set Integrity Pending state when it begins a load operation on a table with constraints.

Load in Progress
This is a transient state that is only in effect during a load operation. For information about bringing a table out of Load in Progress state if your load operation has failed or was interrupted, see the section on pending states after a load operation in the *Related links* section. See also Load in Progress table space state.

Load Pending
A load operation has been active on this table but has been aborted before the data could be committed. Issue a LOAD TERMINATE, LOAD RESTART, or LOAD REPLACE command to bring the table out of this state.

Read Access Only
A table is in this state during a load operation if the ALLOW READ ACCESS option was specified. Read Access Only is a transient state that allows other applications and utilities to have read access to data that existed prior to the load operation.

Reorg Pending
A REORG command recommended ALTER TABLE statement has been executed on the table. A classic REORG must be performed before the table is accessible again.

Unavailable
The table is unavailable. The table can only be dropped or restored from a backup. Rolling forward through a non-recoverable load operation will place a table in the unavailable state.

Not Load Restartable
The table is in a partially loaded state that will not allow a load restart operation. The table will also be in load pending state. Issue a LOAD TERMINATE or a LOAD REPLACE command to bring the table out of the not load restartable state. A table is placed in not load restartable state when a rollforward operation is performed after a failed load operation that has not been successfully restarted or terminated, or when a restore operation is performed from an online backup that was taken while the table was in load in progress or load pending state. In either case, the information required for a load restart operation is unreliable, and the not load restartable state prevents a load restart operation from taking place.

Unknown
The LOAD QUERY command is unable to determine the table state.

There are currently at least 25 table or table space states supported by the IBM DB2 database product. These states are used to control access to data under certain circumstances, or to elicit specific user actions, when required, to protect the integrity of the database. Most of them result from events related to the operation of one of the DB2 utilities, such as the load utility, or the backup and restore utilities.

Although dependent table spaces are no longer quiesced (a quiesce is a persistent lock) prior to a load operation, the Load in Progress table space state prevents the backup of dependent tables during a load operation. The Load in Progress table space state is different from the Load in Progress table state: All load operations use the Load in Progress table state, but load operations (against a recoverable database) with the COPY NO option specified also use the Load in Progress table space state.

The following table describes each of the supported table states. The table also provides you with working examples that show you exactly how to interpret and respond to states that you might encounter while administering your database. The examples are taken from command scripts that were run on AIX; you can copy, paste and run them yourself. If you are running the DB2 database product on a system that is not UNIX, ensure that any path names are in the correct format for your system. Most of the examples are based on tables in the SAMPLE database that comes with the DB2 database product. A few examples require scenarios that are not part of the SAMPLE database, but you can use a connection to the SAMPLE database as a starting point.

Table 47. Supported table states

State	Examples

Load Pending	Given load input file staffdata.del with a substantial amount of data (for example, 20000 or more records), create a small table space that contains the target table of the load operation, a new table called NEWSTAFF:
	`connect to sample;`
	`create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000 /SQL00001/ts1c1' 256);`
	`create table newstaff like staff in ts1;`
	`load from staffdata.del of del insert into newstaff;`
	`load query table newstaff;`
	`load from staffdata.del of del terminate into newstaff;`
	`load query table newstaff;`
	`connect reset;`
	
	Information returned by the LOAD QUERY command shows that the NEWSTAFF table is in Load Pending state; after a load terminate operation, the table is in Normal state.

Load in Progress	Given load input file staffdata.del with a substantial amount of data (for example, 20000 or more records):
	`connect to sample;`
	`create table newstaff like staff;`
	`load from staffdata.del of del insert into newstaff;`
	
	While the load operation is running, execute the following script from another session:
	`connect to sample;`
	`load query table newstaff;`
	`connect reset;`
	
	Information returned by the LOAD QUERY command shows that the NEWSTAFF table is in Load in Progress state.

Normal	`connect to sample;`
	`create table newstaff like staff;`
	`load query table newstaff;`
	
	Information returned by the LOAD QUERY command shows that the NEWSTAFF table is in Normal state.

Table 47. Supported table states (continued)

State	Examples

Not Load Restartable	Given load input file staffdata.del with a substantial amount of data (for example, 20000 or more records):
	`update db cfg for sample using logretain recovery;`
	`backup db sample;`
	`connect to sample;`
	`create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000 /SQL00001/ts1c1' 256);`
	`create table newstaff like staff in ts1;`
	`connect reset;`
	`backup db sample;`
	
	The timestamp for this backup image is: 20040629205935
	`connect to sample;`
	`load from staffdata.del of del insert into newstaff copy yes to /home/melnyk/backups;`
	`connect reset;`
	`restore db sample taken at 20040629205935;`
	`rollforward db sample to end of logs and stop;`
	`connect to sample;`
	`load query table newstaff;`
	`connect reset;`
	
	Information returned by the LOAD QUERY command shows that the NEWSTAFF table is in Not Load Restartable and Load Pending state.
	`connect to sample;`
	`load from staffdata.del of del terminate into newstaff copy yes to /home/melnyk/backups;`
	`load query table newstaff;`
	`connect reset;`
	
	Information returned by the LOAD QUERY command shows that the NEWSTAFF table is now in Normal state.

Read Access Only	Given load input file staffdata.del with a substantial amount of data (for example, 20000 or more records):
	`connect to sample;`
	`export to st_data.del of del select * from staff;`
	`create table newstaff like staff;`
	`import from st_data.del of del insert into newstaff;`
	`load from staffdata.del of del insert into newstaff allow read access;`
	
	While the load operation is running, execute the following script from another session:
	`connect to sample;`
	`load query table newstaff;`
	`select * from newstaff;`
	`connect reset;`
	
	Information returned by the LOAD QUERY command shows that the NEWSTAFF table is in Read Access Only and Load in Progress state. The query returns only the exported contents of the STAFF table, data that existed in the NEWSTAFF table prior to the load operation.

Set Integrity Pending	Given load input file staff_data.del with content:
	
	11,"Melnyk",20,"Sales",10,70000,15000:
	`connect to sample;`
	`alter table staff add constraint max_salary check (100000 - salary > 0);`
	`load from staff_data.del of del insert into staff;`
	`load query table staff;`
	
	Information returned by the LOAD QUERY command shows that the STAFF table is in Set Integrity Pending state.

Table 47. Supported table states (continued)

State	Examples

Unavailable	Given load input file staff_data.del with content:
	
	11,"Melnyk",20,"Sales",10,70000,15000:
	`update db cfg for sample using logretain recovery;`
	`backup db sample;`
	
	The timestamp for this backup image is: 20040629182012
	`connect to sample;`
	`load from staff_data.del of del insert into staff nonrecoverable;`
	`connect reset;`
	`restore db sample taken at 20040629182012;`
	`rollforward db sample to end of logs and stop;`
	`connect to sample;`
	`load query table staff;`
	`connect reset;`
	
	Information returned by the LOAD QUERY command shows that the STAFF table is in Unavailable state.

For additional information about table states, see the *Related links* section.

The progress of a load operation can also be monitored with the LIST UTILITIES command.

LIST TABLESPACES

Lists table spaces and information about table spaces for the current database.

Information displayed by this command is also available in the table space snapshot.

Scope

This command returns information only for the database partition on which it is executed.

Authorization

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- LOAD authority

Required connection

Database

Command syntax

Command parameters

SHOW DETAIL

If this option is not specified, only the following basic information about each table space is provided:

- Table space ID
- Name
- Type (system managed space or database managed space)
- Contents (any data, long or index data, or temporary data)
- State, a hexadecimal value indicating the current table space state. The externally visible state of a table space is composed of the hexadecimal sum of certain state values. For example, if the state is "quiesced: EXCLUSIVE" and "Load pending", the value is `0x0004` + `0x0008`, which is `0x000c`. The db2tbst (Get Tablespace State) command can be used to obtain the table space state associated with a given hexadecimal value. Following are the bit definitions listed in `sqlutil.h`:

```
0x0          Normal
0x1          Quiesced: SHARE
0x2          Quiesced: UPDATE
0x4          Quiesced: EXCLUSIVE
0x8          Load pending
0x10         Delete pending
0x20         Backup pending
0x40         Roll forward in progress
0x80         Roll forward pending
0x100        Restore pending
0x100        Recovery pending (not used)
0x200        Disable pending
0x400        Reorg in progress
0x800        Backup in progress
0x1000       Storage must be defined
0x2000       Restore in progress
0x4000       Offline and not accessible
0x8000       Drop pending
0x20000      Load in progress
0x2000000    Storage may be defined
0x4000000    StorDef is in 'final' state
0x8000000    StorDef was change prior to roll forward
0x10000000   DMS rebalance in progress
0x20000000   Table space deletion in progress
0x40000000   Table space creation in progress
```

If this option is specified, the following additional information about each table space is provided:

- Total number of pages
- Number of usable pages
- Number of used pages
- Number of free pages
- High water mark (in pages)
- Page size (in bytes)
- Extent size (in pages)
- Prefetch size (in pages)

- Number of containers
- Minimum recovery time (displayed only if not zero)
- State change table space ID (displayed only if the table space state is ″load pending″ or ″delete pending″)
- State change object ID (displayed only if the table space state is ″load pending″ or ″delete pending″)
- Number of quiescers (displayed only if the table space state is ″quiesced: SHARE″, ″quiesced: UPDATE″, or ″quiesced: EXCLUSIVE″)
- Table space ID and object ID for each quiescer (displayed only if the number of quiescers is greater than zero).

Examples

The following are two sample outputs from LIST TABLESPACES SHOW DETAIL.

```
          Tablespaces for Current Database
Tablespace ID                        = 0
Name                                 = SYSCATSPACE
Type                                 = Database managed space
Contents                             = Any data
State                                = 0x0000
  Detailed explanation:
    Normal
Total pages                          = 895
Useable pages                        = 895
Used pages                           = 895
Free pages                           = Not applicable
High water mark (pages)              = Not applicable
Page size (bytes)                    = 4096
Extent size (pages)                  = 32
Prefetch size (pages)                = 32
Number of containers                 = 1

Tablespace ID                        = 1
Name                                 = TEMPSPACE1
Type                                 = System managed space
Contents                             = Temporary data
State                                = 0x0000
  Detailed explanation:
    Normal
Total pages                          = 1
Useable pages                        = 1
Used pages                           = 1
Free pages                           = Not applicable
High water mark (pages)              = Not applicable
Page size (bytes)                    = 4096
Extent size (pages)                  = 32
Prefetch size (pages)                = 32
Number of containers                 = 1

Tablespace ID                        = 2
Name                                 = USERSPACE1
Type                                 = Database managed space
Contents                             = Any data
State                                = 0x000c
  Detailed explanation:
    Quiesced: EXCLUSIVE
    Load pending
Total pages                          = 337
Useable pages                        = 337
Used pages                           = 337
Free pages                           = Not applicable
High water mark (pages)              = Not applicable
```

```
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
 State change tablespace ID           = 2
 State change object ID               = 3
 Number of quiescers                  = 1
   Quiescer 1:
     Tablespace ID                    = 2
     Object ID                        = 3
DB21011I  In a partitioned database server environment, only the table spaces
on the current node are listed.

           Tablespaces for Current Database
 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1200
 Useable pages                        = 1200
 Used pages                           = 1200
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1

 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = Temporary data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1

Tablespace ID                         = 2
 Name                                 = USERSPACE1
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1

Tablespace ID                         = 3
 Name                                 = DMS8K
```

```
 Type                                = Database managed space
 Contents                            = Any data
 State                               = 0x0000
   Detailed explanation:
     Normal
 Total pages                         = 2000
 Useable pages                       = 1952
 Used pages                          = 96
 Free pages                          = 1856
 High water mark (pages)             = 96
 Page size (bytes)                   = 8192
 Extent size (pages)                 = 32
 Prefetch size (pages)               = 32
 Number of containers                = 2

Tablespace ID                        = 4
 Name                                = TEMP8K
 Type                                = System managed space
 Contents                            = Temporary data
 State                               = 0x0000
   Detailed explanation:
     Normal
 Total pages                         = 1
 Useable pages                       = 1
 Used pages                          = 1
 Free pages                          = Not applicable
 High water mark (pages)             = Not applicable
 Page size (bytes)                   = 8192
 Extent size (pages)                 = 32
 Prefetch size (pages)               = 32
 Number of containers                = 1
DB21011I  In a partitioned database server environment, only the table spaces
on the current node are listed.
```

Usage notes

In a partitioned database environment, this command does not return all the table spaces in the database. To obtain a list of all the table spaces, query SYSCAT.TABLESPACES.

During a table space rebalance, the number of usable pages includes pages for the newly added container, but these new pages are not reflected in the number of free pages until the rebalance is complete. When a table space rebalance is not in progress, the number of used pages plus the number of free pages equals the number of usable pages.

There are currently at least 25 table or table space states supported by the IBM DB2 database product. These states are used to control access to data under certain circumstances, or to elicit specific user actions, when required, to protect the integrity of the database. Most of them result from events related to the operation of one of the DB2 utilities, such as the load utility, or the backup and restore utilities.

The following table describes each of the supported table space states. The table also provides you with working examples that show you exactly how to interpret and respond to states that you might encounter while administering your database. The examples are taken from command scripts that were run on AIX; you can copy, paste and run them yourself. If you are running the DB2 database product on a system that is not UNIX, ensure that any path names are in the correct format for your system. Most of the examples are based on tables in the SAMPLE database

that comes with the DB2 database product. A few examples require scenarios that are not part of the SAMPLE database, but you can use a connection to the SAMPLE database as a starting point.

Table 48. Supported table space states

State	Hexadecimal state value	Description	Examples

Backup Pending	0x20	A table space is in this state after a point-in-time table space rollforward operation, or after a load operation (against a recoverable database) that specifies the COPY NO option. The table space (or, alternatively, the entire database) must be backed up before the table space can be used. If the table space is not backed up, tables within that table space can be queried, but not updated.
		Note: A database must also be backed up immediately after it is enabled for rollforward recovery. A database is recoverable if the **logretain** database configuration parameter is set to RECOVERY, or the **userexit** database configuration parameter is set to YES. You cannot activate or connect to such a database until it has been backed up, at which time the value of the **backup_pending** informational database configuration parameter is set to NO.
			1. Given load input file staff_data.del with content:
			11,"Melnyk",20,"Sales",10,70000,15000:
			`update db cfg for sample using logretain recovery;`
			`backup db sample;`
			`connect to sample;`
			`load from staff_data.del of del messages load.msg`
			`insert into staff copy no;`
			`update staff set salary = 69000 where id = 11;`
			2.
			`update db cfg for sample using logretain recovery;`
			`connect to sample;`

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Backup in Progress	0x800	This is a transient state that is only in effect during a backup operation.	Issue an online BACKUP DATABASE command:
			`backup db sample online;`
			
			While the backup operation is running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Backup in Progress state.

DMS Rebalance in Progress	0x10000000	This is a transient state that is only in effect during a data rebalancing operation. When new containers are added to a table space that is defined as database managed space (DMS), or existing containers are extended, a rebalancing of the table space data might occur. *Rebalancing* is the process of moving table space extents from one location to another in an attempt to keep the data striped. An *extent* is a unit of container space (measured in pages), and a stripe is a layer of extents *across the set of containers* for a table space.	Given load input file staffdata.del with a substantial amount of data (for example, 20000 or more records):
			`connect to sample;`
			`create tablespace ts1 managed by database using`
			`(file '/home/melnyk/melnyk/NODE0000/SQL00001`
			`/ts1c1' 1024);`
			`create table newstaff like staff in ts1;`
			`load from staffdata.del of del insert into newstaff`
			`nonrecoverable;`
			`alter tablespace ts1 add (file '/home/melnyk/melnyk`
			`/NODE0000/SQL00001/ts1c2' 1024);`
			`list tablespaces;`
			`connect reset;`
			
			Information returned for TS1 shows that this table space is in DMS Rebalance in Progress state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples
Disable Pending	0x200	A table space may be in this state during a database rollforward operation and should no longer be in this state by the end of the rollforward operation. The state is triggered by conditions that result from a table space going offline and compensation log records for a transaction not being written. The appearance and subsequent disappearance of this table space state is transparent to users.	An example illustrating this table space state is beyond the scope of this document.
Drop Pending	0x8000	A table space is in this state if one or more of its containers is found to have a problem during a database restart operation. (A database must be restarted if the previous session with this database terminated abnormally, such as during a power failure, for example.) If a table space is in Drop Pending state, it will not be available, and can only be dropped.	An example illustrating this table space state is beyond the scope of this document.

Load in Progress	0x20000	This is a transient state that is only in effect during a load operation (against a recoverable database) that specifies the COPY NO option. See also Load in Progress table state.	Given load input file staffdata.del with a substantial amount of data (for example, 20000 or more records):
			`update db cfg for sample using logretain recovery;`
			`backup db sample;`
			`connect to sample;`
			`create table newstaff like staff;`
			`load from staffdata.del of del insert into newstaff copy no;`
			`connect reset;`
			
			While the load operation is running, execute the following script from another session:
			`connect to sample;`
			`list tablespaces;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Load in Progress (and Backup Pending) state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Normal	0x0	A table space is in Normal state if it is not in any of the other (abnormal) table space states. Normal state is the initial state of a table space after it is created.	`connect to sample;`
			`create tablespace ts1 managed by database using`
			`(file '/home/melnyk/melnyk/NODE0000/SQL00001`
			`/tsc1' 1024);`
			`list tablespaces show detail;`

Offline and Not Accessible	0x4000	A table space is in this state if there is a problem with one or more of its containers. A container might be inadvertently renamed, moved, or damaged. After the problem has been rectified, and the containers that are associated with the table space are accessible again, this abnormal state can be removed by disconnecting all applications from the database and then reconnecting to the database. Alternatively, you can issue an ALTER TABLESPACE statement, specifying the SWITCH ONLINE clause, to remove the Offline and Not Accessible state from the table space without disconnecting other applications from the database.	`connect to sample;`
			`create tablespace ts1 managed by database using`
			`(file '/home/melnyk/melnyk/NODE0000/SQL00001`
			`/tsc1' 1024);`
			`alter tablespace ts1 add (file '/home/melnyk/melnyk`
			`/NODE0000/SQL00001/tsc2' 1024);`
			`export to st_data.del of del select * from staff;`
			`create table stafftemp like staff in ts1;`
			`import from st_data.del of del insert into stafftemp;`
			`connect reset;`
			
			Rename table space container tsc1 to tsc3 and then try to query the STAFFTEMP table:
			`connect to sample;`
			`select * from stafftemp;`
			
			The query returns SQL0290N (table space access is not allowed), and the LIST TABLESPACES command returns a state value of 0x4000 (Offline and Not Accessible) for TS1. Rename table space container tsc3 back to tsc1. This time the query runs successfully.

Quiesced Exclusive	0x4	A table space is in this state when the application that invokes the table space quiesce function has exclusive (read or write) access to the table space. You can put a table space in Quiesced Exclusive state explicitly by issuing the QUIESCE TABLESPACES FOR TABLE command.	Ensure that the table space state is Normal before setting it to Quiesced Exclusive.
			`connect to sample;`
			`quiesce tablespaces for table staff reset;`
			`quiesce tablespaces for table staff exclusive;`
			`connect reset;`
			
			Execute the following script from another session:
			`connect to sample;`
			`select * from staff where id=60;`
			`update staff set salary=50000 where id=60;`
			`list tablespaces;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Quiesced Exclusive state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Quiesced Share	0x1	A table space is in this state when both the application that invokes the table space quiesce function and concurrent applications have read (but not write) access to the table space. You can put a table space in Quiesced Share state explicitly by issuing the QUIESCE TABLESPACES FOR TABLE command.	Ensure that the table space state is Normal before setting it to Quiesced Share.
			`connect to sample;`
			`quiesce tablespaces for table staff reset;`
			`quiesce tablespaces for table staff share;`
			`connect reset;`
			
			Execute the following script from another session:
			`connect to sample;`
			`select * from staff where id=40;`
			`update staff set salary=50000 where id=40;`
			`list tablespaces;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Quiesced Share state.

Quiesced Update	0x2	A table space is in this state when the application that invokes the table space quiesce function has exclusive write access to the table space. You can put a table space in Quiesced Update state explicitly by issuing the QUIESCE TABLESPACES FOR TABLE command.	Ensure that the table space state is Normal before setting it to Quiesced Update.
			`connect to sample;`
			`quiesce tablespaces for table staff reset;`
			`quiesce tablespaces for table staff intent to update;`
			`connect reset;`
			
			Execute the following script from another session:
			`connect to sample;`
			`select * from staff where id=50;`
			`update staff set salary=50000 where id=50;`
			`list tablespaces;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Quiesced Update state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Reorg in Progress	0x400	This is a transient state that is only in effect during a reorg operation.	Issue a REORG TABLE command:
			`connect to sample;`
			`reorg table staff;`
			`connect reset;`
			
			While the reorg operation is running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Reorg in Progress state.
			Note: Table reorganization operations involving the SAMPLE database are likely to complete in a short period of time and, as a result, it may be difficult to observe the Reorg in Progress state using this approach.

Restore Pending	0x100	Table spaces for a database are in this state after the first part of a redirected restore operation (that is, before the SET TABLESPACE CONTAINERS command is issued). The table space (or the entire database) must be restored before the table space can be used. You cannot connect to the database until the restore operation has been successfully completed, at which time the value of the **restore_pending** informational database configuration parameter is set to NO.	When the first part of the redirected restore operation in Storage May be Defined completes, all of the table spaces are in Restore Pending state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Restore in Progress	0x2000	This is a transient state that is only in effect during a restore operation.	`update db cfg for sample using logretain recovery;`
			`backup db sample;`
			`backup db sample tablespace (userspace1);`
			
			The timestamp for this backup image is:
			
			20040611174124
			
			`restore db sample tablespace (userspace1) online taken at 20040611174124;`
			
			While the restore operation is running, execute the following script from another session:
			
			`connect to sample;`
			
			1.
			
			`list tablespaces show detail;`
			
			or
			
			2.
			
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for USERSPACE1 shows that this table space is in Restore in Progress state.

Roll Forward Pending	0x80	A table space is in this state after a restore operation against a recoverable database. The table space (or the entire database) must be rolled forward before the table space can be used. A database is recoverable if the **logretain** database configuration parameter is set to RECOVERY, or the **userexit** database configuration parameter is set to YES. You cannot activate or connect to the database until a rollforward operation has been successfully completed, at which time the value of the **rollfwd_pending** informational database configuration parameter is set to NO.	When the online table space restore operation in Restore in Progress completes, the table space USERSPACE1 is in Roll Forward Pending state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Roll Forward in Progress	0x40	This is a transient state that is only in effect during a rollforward operation.	Given load input file staffdata.del with a substantial amount of data (for example, 20000 or more records):
			`update db cfg for sample using logretain recovery;`
			`backup db sample;`
			`connect to sample;`
			`create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001 /ts1c1' 1024);`
			`create table newstaff like staff in ts1;`
			`connect reset;`
			`backup db sample tablespace (ts1) online;`
			
			The timestamp for this backup image is:
			
			20040630000715
			
			`connect to sample;`
			`load from staffdata.del of del insert into newstaff copy yes to /home/melnyk/backups;`
			`connect reset;`
			`restore db sample tablespace (ts1) online taken at 20040630000715;`
			`rollforward db sample to end of logs and stop tablespace (ts1) online;`
			
			While the rollforward operation is running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for TS1 shows that this table space is in Roll Forward in Progress state.

Storage May be Defined	0x2000000	Table spaces for a database are in this state after the first part of a redirected restore operation (that is, before the SET TABLESPACE CONTAINERS command is issued). This allows you to redefine the containers, if you wish.	`backup db sample;`
			
			Assuming that the timestamp for this backup image is 20040613204955:
			`restore db sample taken at 20040613204955 redirect;`
			`list tablespaces;`
			
			Information returned by the LIST TABLESPACES command shows that all of the table spaces are in Storage May be Defined and Restore Pending state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Storage Must be Defined	0x1000	Table spaces for a database are in this state during a redirected restore operation to a new database if the set table space containers phase is omitted or if, during the set table space containers phase, the specified containers cannot be acquired. The latter can occur if, for example, an invalid path name has been specified, or there is insufficient disk space.	`backup db sample;`
			
			Assuming that the timestamp for this backup image is 20040613204955:
			`restore db sample taken at 20040613204955 into mydb redirect;`
			`set tablespace containers for 2 using (path 'ts2c1');`
			`list tablespaces;`
			
			Information returned by the LIST TABLESPACES command shows that table space SYSCATSPACE and table space TEMPSPACE1 are in Storage Must be Defined, Storage May be Defined, and Restore Pending state. Storage Must be Defined state takes precedence over Storage May be Defined state.

Table Space Creation in Progress	0x40000000	This is a transient state that is only in effect during a create table space operation.	`connect to sample;`
			`create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc1' 1024);`
			`create tablespace ts2 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc2' 1024);`
			`create tablespace ts3 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001/tsc3' 1024);`
			
			While the create table space operations are running, execute the following script from another session:
			`connect to sample;`
			
			1.
			`list tablespaces show detail;`
			
			or
			
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Creation in Progress state.

Table 48. Supported table space states (continued)

State	Hexadecimal state value	Description	Examples

Table Space Deletion in Progress	0x20000000	This is a transient state that is only in effect during a delete table space operation.	`connect to sample;`
			`create tablespace ts1 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001 /tsc1' 1024);`
			`create tablespace ts2 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001 /tsc2' 1024);`
			`create tablespace ts3 managed by database using (file '/home/melnyk/melnyk/NODE0000/SQL00001 /tsc3' 1024);`
			`drop tablespace ts1;`
			`drop tablespace ts2;`
			`drop tablespace ts3;`
			
			While the drop table space operations are running, execute the following script from another session:
			`connect to sample;`
			1.
			`list tablespaces show detail;`
			or
			2.
			`get snapshot for tablespaces on sample;`
			`connect reset;`
			
			Information returned for TS1, TS2, and TS3 shows that these table spaces are in Table Space Deletion in Progress state.

For additional information about table space states, see the *Related links* section.

Chapter 5. Other data movement options

Moving data with DB2 Connect

If you are working in a complex environment in which you need to move data between a host database system and a workstation, you can use DB2 Connect, the gateway for data transfer between the host and the workstation (see Figure 17).

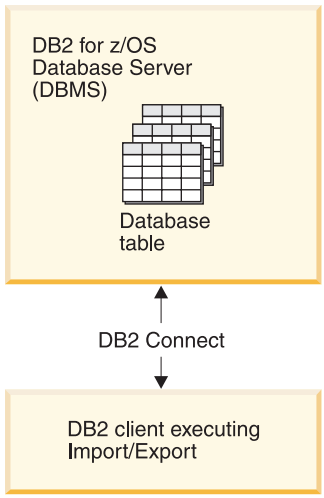


Figure 17. Import/Export through DB2 Connect

The DB2 export and import utilities allow you to move data from a host or System i server database to a file on the DB2 Connect workstation, and the reverse. You can then use the data with any other application or relational database management system that supports this export or import format. For example, you can export data from a host or System i server database into a PC/IXF file, and then import it into a DB2 for Windows database.

You can perform export and import operations from a database client or from the DB2 Connect workstation.

Note:

1. The data to be exported or imported must comply with the size and data type restrictions that are applicable to both databases.
2. To improve import performance, you can use compound queries. Specify the `compound` file type modifier in the import utility to group a specified number of query statements into a block. This can reduce network overhead and improve response time.

With DB2 Connect, export and import operations must meet the following conditions:

- The file type must be PC/IXF.
- A target table with attributes that are compatible with the data must be created on the target server before you can import to it. The db2look utility can be used

© Copyright IBM Corp. 1993, 2008

to get the attributes of the source table. Import through DB2 Connect cannot create a table, because INSERT is the only supported option.

If any of these conditions is not met, the operation fails, and an error message is returned.

Note: Index definitions are not stored on export or used on import.

If you export or import mixed data (columns containing both single-byte and double-byte data), consider the following:

- On systems that store data in EBCDIC (MVS, OS/390, OS/400, VM, and VSE), shift-out and shift-in characters mark the start and the end of double-byte data. When you define column lengths for your database tables, be sure to allow enough room for these characters.
- Variable-length character columns are recommended, unless the column data has a consistent pattern.

Moving Data from a workstation to a host server

To move data to a host or System i server database:

1. Export the data from a DB2 table to a PC/IXF file
2. Using the INSERT option, import the PC/IXF file into a compatible table in the host server database.

To move data from a host server database to a workstation:

1. Export the data from the host server database table to a PC/IXF file.
2. Import the PC/IXF file into a DB2 table.

Example

The following example illustrates how to move data from a workstation to a host or System i server database.

Export the data into an external IXF format by issuing the following command:

```
db2 export to staff.ixf of ixf select * from userid.staff
```

Issue the following command to establish a DRDA connection to the target DB2 database:

```
db2 connect to cbc664 user admin using xxx
```

If it doesn't already exit, create the target table on the target DB2 database instance:

```
CREATE TABLE mydb.staff (ID SMALLINT NOT NULL, NAME VARCHAR(9),
       DEPT SMALLINT, JOB CHAR(5), YEARS SMALLINT, SALARY DECIMAL(7,2),
       COMM DECIMAL(7,2))
```

To import the data issue the following command:

```
db2 import from staff.ixf of ixf insert into mydb.staff
```

Each row of data will be read from the file in IXF format, and an SQL INSERT statement will be issued to insert the row into table mydb.staff. Single rows will continue to be inserted until all of the data has been moved to the target table.

Detailed information is available in "Moving Data Across the DB2 Family," an IBM Redbooks™ publication. This Redbooks publication can be found at the following URL: http://www.redbooks.ibm.com/redbooks/SG246905.

The IBM Replication Tools by Component

IBM offers two primary replication solutions: Q replication and SQL replication.

The primary components of Q replication are the Q Capture program and the Q Apply program. The primary components of SQL replication are the Capture program and Apply program. Both types of replication share the Replication Alert Monitor tool. You can set up and administer these replication components using the Replication Center and the ASNCLP command-line program.

The following list briefly summarizes these replication components:

Q Capture program

Reads the DB2 recovery log looking for changes to DB2 source tables and translates committed source data into WebSphere® MQ messages that can be published in XML format to a subscribing application, or replicated in a compact format to the Q Apply program.

Q Apply program

Takes WebSphere MQ messages from a queue, transforms the messages into SQL statements, and updates a target table or stored procedure. Supported targets include DB2 databases or subsystems and Oracle, Sybase, Informix® and Microsoft® SQL Server databases that are accessed through federated server nicknames.

Capture program

Reads the DB2 recovery log for changes made to registered source tables or views and then stages committed transactional data in relational tables called change-data (CD) tables, where they are stored until the target system is ready to copy them. SQL replication also provides Capture triggers that populate a staging table called a consistent-change-data (CCD) table with records of changes to non-DB2 source tables.

Apply program

Reads data from staging tables and makes the appropriate changes to targets. For non-DB2 data sources, the Apply program reads the CCD table through that table's nickname on the federated database and makes the appropriate changes to the target table.

Replication Alert Monitor

A utility that checks the health of the Q Capture, Q Apply, Capture, and Apply programs. It checks for situations in which a program terminates, issues a warning or error message, reaches a threshold for a specified value, or performs a certain action, and then issues notifications to an email server, pager, or the z/OS console.

Use the Replication Center to:

- Define registrations, subscriptions, publications, queue maps, alert conditions, and other objects.

- Start, stop, suspend, resume, and reinitialize the replication programs.
- Specify the timing of automated copying.
- Specify SQL enhancements to the data.
- Define relationships between the source and the target tables.

Copying schemas

The db2move utility and the ADMIN_COPY_SCHEMA procedure allow you to quickly make copies of a database schema. Once a model schema is established, you can use it as a template for creating new versions.

Use the ADMIN_COPY_SCHEMA procedure to copy a single schema within the same database or the db2move utility with the `-co COPY` action to copy a single schema or multiple schemas from a source database to a target database. Most database objects from the source schema are copied to the target database under the new schema.

Troubleshooting tips

Both the ADMIN_COPY_SCHEMA procedure and the db2move utility invoke the LOAD command. While the load is processing, the table spaces wherein the database target objects reside are put into backup pending state.

ADMIN_COPY_SCHEMA procedure

Using this procedure with the COPYNO option places the table spaces wherein the target object resides into backup pending state, as described in the note above. To get the table space out of the set integrity pending state, this procedure issues a SET INTEGRITY statement. In situations where a target table object has referential constraints defined, the target table is also placed in the set integrity pending state. Because the table spaces are already in backup pending state, the ADMIN_COPY_SCHEMA procedure's attempt to issue a SET INTEGRITY statement will fail.

To resolve this situation, issue a BACKUP DATABASE command to get the affected table spaces out of backup pending state. Next, look at the **Statement_text** column of the error table generated by this procedure to find a list of tables in the set integrity pending state. Then issue the SET INTEGRITY statement for each of the tables listed to take each table out of the set integrity pending state.

db2move utility

This utility attempts to copy all allowable schema objects with the exception of the following types:

- table hierarchy
- staging tables (not supported by the load utility in multiple partition database environments)
- jars (Java™ routine archives)
- nicknames
- packages
- view hierarchies
- object privileges (All new objects are created with default authorizations)
- statistics (New objects do not contain statistics information)
- index extensions (user-defined structured type related)
- user-defined structured types and their transform functions

Unsupported type errors
If an object of one of the unsupported types is detected in the source schema, an entry is logged to an error file, indicating that an unsupported object type is detected. The COPY operation will still succeed–the logged entry is meant to inform you of objects not copied by this operation.

Objects not coupled with schemas
Objects that are not coupled with a schema, such as table spaces and event monitors, are not operated on during a copy schema operation. You should create them on the target database before the copy schema operation is invoked.

Replicated tables
When copying a replicated table, the new copy of the table is not enabled for replication. The table is re-created as a regular table.

Different instances
The source database must be cataloged if it does not reside in the same instance as the target database.

SCHEMA_MAP option
When using the SCHEMA_MAP option to specify a different schema name on the target database, the copy schema operation will perform only minimal parsing of the object definition statements to replace the original schema name with the new schema name. For example, any instances of the original schema that appear inside the contents of an SQL procedure are not replaced with the new schema name. Thus the copy schema operation may fail to recreate these objects. You can use the DDL in the error file to manually recreate these failed objects after the copy operation completes.

Interdependencies between objects
The copy schema operation attempts to recreate objects in an order that satisfies the interdependencies between these objects. For example, if a table T1 contains a column that references a user-defined function U1, then it will recreate U1 before recreating T1. However, dependency information for procedures is not readily available in the catalogs, so when recreating procedures, the copy schema operation will first attempt to recreate all procedures, then retry to recreate those that failed (on the assumption that if they depended on a procedure that was successfully created during the previous attempt, then during a subsequent attempt they will be recreated successfully). The operation will continually try to recreate these failed procedures as long as it is able to successfully recreate one or more during a subsequent attempt. During every attempt at recreating a procedure, an error (and DDL) is logged into the error file. You might see many entries in the error file for the same procedures, but these procedures might have even been successfully recreated during a subsequent attempt. You should query the SYSCAT.PROCEDURES table upon completion of the copy schema operation to determine if these procedures listed in the error file were successfully recreated.

For more information, see the ADMIN_COPY_SCHEMA procedure and the db2move utility.

Examples of schema copy using the db2move utility

Use the db2move utility with the -co COPY action to copy one or more schemas from a source database to a target database. Once a model schema is established, you can use it as a template for creating new versions.

Example 1: Using the -c COPY options

The following example of the db2move -co COPY options copies the schema BAR and renames it FOO from the sample database to the target database:

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" -u userid -p password
```

The new (target) schema objects are created using the same object names as the objects in the source schema, but with the target schema qualifier. It is possible to create copies of tables with or without the data from the source table. The source and target databases can be on different systems.

Example 2: Specifying table space name mappings during the COPY operation

The following example shows how to specify specific table space name mappings to be used instead of the table spaces from the source system during a db2move COPY operation. You can specify the SYS_ANY keyword to indicate that the target table space should be chosen using the default table space selection algorithm. In this case, the db2move utility chooses any available table space to be used as the target:

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" -u userid -p password
```

The SYS_ANY keyword can be used for all table spaces, or you can specify specific mappings for some table spaces, and the default table space selection algorithm for the remaining:

```
db2move sample COPY -sn BAR -co target_db  target schema_map "
  ((BAR,FOO))" tablespace_map "((TS1, TS2),(TS3, TS4), SYS_ANY)"
  -u userid -p password
```

This indicates that table space TS1 is mapped to TS2, TS3 is mapped to TS4, but the remaining table spaces use a default table space selection algorithm.

Example 3: Changing the object owners after the COPY operation

You can change the owner of each new object created in the target schema after a successful COPY. The default owner of the target objects is the connect user. If this option is specified, ownership is transferred to a new owner as demonstrated:

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" owner jrichards
  -u userid -p password
```

The new owner of the target objects is jrichards.

The db2move utility must be invoked on the target system if source and target schemas reside on different systems. For copying schemas from one database to another, this action requires a list of schema names to be copied from a source database, separated by commas, and a target database name.

To copy a schema, issue db2move from an OS command prompt as follows:

```
db2move <dbname> COPY -co <COPY- options>
 -u <userid> -p <password>
```

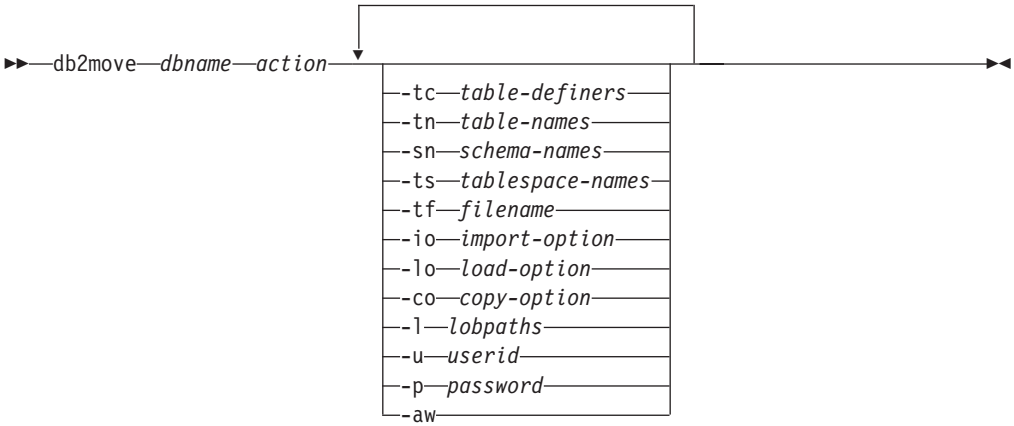
db2move - Database movement tool

This tool, when used in the EXPORT/IMPORT/LOAD mode, facilitates the movement of large numbers of tables between DB2 databases located on workstations. The tool queries the system catalog tables for a particular database and compiles a list of all user tables. It then exports these tables in PC/IXF format. The PC/IXF files can be imported or loaded to another local DB2 database on the same system, or can be transferred to another workstation platform and imported or loaded to a DB2 database on that platform. Tables with structured type columns are not moved when this tool is used. When used in the COPY mode, this tool facilitates the duplication of a schema.

Authorization

This tool calls the DB2 export, import, and load APIs, depending on the action requested by the user. Therefore, the requesting user ID must have the correct authorization required by those APIs, or the request will fail.

Command syntax



Command parameters

dbname
Name of the database.

action Must be one of:

EXPORT
Exports all tables that meet the filtering criteria in `options`. If no `options` are specified, exports all the tables. Internal staging information is stored in the `db2move.lst` file.

IMPORT
Imports all tables listed in the internal staging file `db2move.lst`. Use the -io option for IMPORT specific actions.

LOAD
Loads all tables listed in the internal staging file `db2move.lst`. Use the -lo option for LOAD specific actions.

COPY Duplicates a schema(s) into a target database. Use the -sn option to

specify one or more schemas. See the -co option for COPY specific options. Use the -tn or -tf option to filter tables in LOAD_ONLY mode.

See below for a list of files that are generated during each action.

-tc *table-definers*
The default is all definers.

This is an EXPORT action only. If specified, only those tables created by the definers listed with this option are exported. If not specified, the default is to use all definers. When specifying multiple definers, they must be separated by commas; no blanks are allowed between definer IDs. This option can be used with the -tn *table-names* option to select the tables for export.

An asterisk (*) can be used as a wildcard character that can be placed anywhere in the string.

-tn *table-names*
The default is all user tables.

This is an EXPORT or COPY action only.

If specified with the EXPORT action, only those tables whose names match those in the specified string are exported. If not specified, the default is to use all user tables. When specifying multiple table names, they must be separated by commas; no blanks are allowed between table names. Table names should be listed unqualified and the -sn option should be used to filter schemas.

For export, an asterisk (*) can be used as a wildcard character that can be placed anywhere in the string.

If specified with the COPY action, the -co "MODE" LOAD_ONLY *copy-option* must also be specified, and only those tables specified will be repopulated on the target database. The table names should be listed with their schema qualifier in the format "schema"."table".

-sn *schema-names*
The default for EXPORT is all schemas (not for COPY).

If specified, only those tables whose schema names match will be exported or copied. If multiple schema names are specified, they must be separated by commas; no blanks are allowed between schema names. Schema names of less than 8 characters are padded to 8 characters in length.

In the case of export: If the asterisk wildcard character (*) is used in the schema names, it will be changed to a percent sign (%) and the table name (with percent sign) will be used in the LIKE predicate of the WHERE clause. If not specified, the default is to use all schemas. If used with the -tn or -tc option, db2move will only act on those tables whose schemas match the specified schema names and whose definers match the specified definers. A schema name `fred` has to be specified `-sn fr*d*` instead of `-sn fr*d` when using an asterisk.

-ts *tablespace-names*
The default is all table spaces.

This is an EXPORT action only. If this option is specified, only those tables that reside in the specified table space will be exported. If the asterisk wildcard character (*) is used in the table space name, it will be changed to

a percent sign (%) and the table name (with percent sign) will be used in the LIKE predicate in the WHERE clause. If the -ts option is not specified, the default is to use all table spaces. If multiple table space names are specified, they must be separated by commas; no blanks are allowed between table space names. Table space names less than 8 characters are padded to 8 characters in length. For example, a table space name `mytb` has to be specified `-ts my*b*` instead of `-sn my*b` when using the asterisk.

-tf *filename*

If specified with EXPORT action, only those tables whose names match exactly those in the specified file are exported. If not specified, the default is to use all user tables. The tables should be listed one per line, and each table should be fully qualified. Wildcard characters are not allowed in the strings. Here is an example of the contents of a file:

```
"SCHEMA1"."TABLE NAME1"
"SCHEMA NAME77"."TABLE155"
```

If specified with the COPY action, the -co ″MODE″ LOAD_ONLY *copy-option* must also be specified, and only those tables specified in the file will be repopulated on the target database. The table names should be listed with their schema qualifier in the format ″schema″.″table″.

-io *import-option*

The default is REPLACE_CREATE. See "IMPORT command options CREATE and REPLACE_CREATE are deprecated" for limitations of import create function.

Valid options are: INSERT, INSERT_UPDATE, REPLACE, CREATE, and REPLACE_CREATE.

-lo *load-option*

The default is INSERT.

Valid options are: INSERT and REPLACE.

-co When the db2move action is COPY, the following -co follow-on options will be available:

"TARGET_DB *db name* **[USER** *userid* **USING** *password***]"**

Allows the user to specify the name of the target database and the user/password. (The source database user/password can be specified using the existing -p and -u options). The `USER/USING` clause is optional. If USER specifies a userid, then the password must either be supplied following the USING clause, or if it's not specified, then db2move will prompt for the password information. The reason for prompting is for security reasons discussed below. TARGET_DB is a mandatory option for the COPY action. The TARGET_DB cannot be the same as the source database. The `ADMIN_COPY_SCHEMA` procedure can be used for copying schemas within the same database. The COPY action requires inputting at least one schema (-sn) or one table (-tn or -tf).

Running multiple db2move commands to copy schemas from one database to another will result in deadlocks. Only one db2move command should be issued at a time. Changes to tables in the source schema during copy processing may mean that the data in the target schema is not identical following a copy.

"MODE"

DDL_AND_LOAD
Creates all supported objects from the source schema, and populates the tables with the source table data. This is the default option.

DDL_ONLY
Creates all supported objects from the source schema, but does not repopulate the tables.

LOAD_ONLY
Loads all specified tables from the source database to the target database. The tables must already exist on the target. The LOAD_ONLY mode requires inputting at least one table using the -tn or -tf option.

This is an optional option that is only used with the COPY action.

"SCHEMA_MAP"
Allows user to rename schema when copying to target. Provides a list of the source-target schema mapping, separated by commas, surrounded by brackets. e.g schema_map ((s1, t1), (s2, t2)). This would mean objects from schema s1 will be copied to schema t1 on the target; objects from schema s2 will be copied to schema t2 on the target. The default, and recommended, target schema name is the source schema name. The reason for this is db2move will not attempt to modify the schema for any qualified objects within object bodies. Therefore, using a different target schema name may lead to problems if there are qualified objects within the object body.

For example:`create view FOO.v1 as 'select c1 from FOO.t1'`

In this case, copy of schema FOO to BAR, v1 will be regenerated as:`create view BAR.v1 as 'select c1 from FOO.t1'`

This will either fail since schema FOO does not exist on the target database, or have an unexpected result due to FOO being different than BAR. Maintaining the same schema name as the source will avoid these issues. If there are cross dependencies between schemas, all inter-dependant schemas must be copied or there may be errors copying the objects with the cross dependencies.

For example:`create view FOO.v1 as 'select c1 from BAR.t1'`

In this case, the copy of v1 will either fail if BAR is not copied as well, or have an unexpected result if BAR on the target is different than BAR from the source. db2move will not attempt to detect cross schema dependencies.

This is an optional option that is only used with the COPY action.

"NONRECOVERABLE"
This option allows the user to override the default behavior of the load to be done with COPY-NO. With the default behavior, the user will be forced to take backups of each table space that was loaded into. When specifying this NONRECOVERABLE keyword, the user will not be forced to take backups of the table spaces immediately. It is, however, highly recommended that the backups be taken as soon as possible to ensure the newly created tables will be properly recoverable. This is an optional option available to the COPY action.

"OWNER"

Allows the user to change the owner of each new object created in the target schema after a successful COPY. The default owner of the target objects will be the connect user; if this option is specified, ownership will be transferred to the new owner. This is an optional option available to the COPY action.

"TABLESPACE_MAP"

The user may specify table space name mappings to be used instead of the table spaces from the source system during a copy. This will be an array of table space mappings surrounded by brackets. For example, `tablespace_map ((TS1, TS2),(TS3, TS4))`. This would mean that all objects from table space TS1 will be copied into table space TS2 on the target database and objects from table space TS3 will be copied into table space TS4 on the target. In the case of `((T1, T2),(T2, T3))`, all objects found in T1 on the source database will be recreated in T2 on the target database and any objects found in T2 on the source database will be recreated in T3 on the target database. The default is to use the same table space name as from the source, in which case, the input mapping for this table space is not necessary. If the specified table space does not exist, the copy of the objects using that table space will fail and be logged in the error file.

The user also has the option of using the `SYS_ANY` keyword to indicate that the target table space should be chosen using the default table space selection algorithm. In this case, db2move will be able to choose any available table space to be used as the target. The `SYS_ANY` keyword can be used for all table spaces, example: `tablespace_map SYS_ANY`. In addition, the user can specify specific mappings for some table spaces, and the default table space selection algorithm for the remaining. For example, `tablespace_map ((TS1, TS2),(TS3, TS4), SYS_ANY)`. This indicates that table space TS1 is mapped to TS2, TS3 is mapped to TS4, but the remaining table spaces will be using a default table space target. The `SYS_ANY` keyword is being used since it's not possible to have a table space starting with "SYS".

This is an optional option available to the COPY action.

-l *lobpaths*

For IMPORT and EXPORT, if this option is specified, it will be also used for XML paths. The default is the current directory.

This option specifies the absolute path names where LOB or XML files are created (as part of EXPORT) or searched for (as part of IMPORT or LOAD). When specifying multiple paths, each must be separated by commas; no blanks are allowed between paths. If multiple paths are specified, EXPORT will use them in round-robin fashion. It will write one LOB document to the first path, one to the second path, and so on up to the last, then back to the first path. The same is true for XML documents. If files are not found in the first path (during IMPORT or LOAD), the second path will be used, and so on.

-u *userid*

The default is the logged on user ID.

Both user ID and password are optional. However, if one is specified, the other must be specified. If the command is run on a client connecting to a remote server, user ID and password should be specified.

-p *password*
The default is the logged on password. Both user ID and password are optional. However, if one is specified, the other must be specified. When the -p option is specified, but the password not supplied, db2move will prompt for the password. This is done for security reasons. Inputting the password through command line creates security issues. For example, a `ps -ef` command would display the password. If, however, db2move is invoked through a script, then the passwords will have to be supplied. If the command is issued on a client connecting to a remote server, user ID and password should be specified.

-aw Allow Warnings. When -aw is not specified, tables that experience warnings during export are not included in the `db2move.lst` file (although that table's .ixf file and .msg file are still generated). In some scenarios (such as data truncation) the user might want to allow such tables to be included in the `db2move.lst` file. Specifying this option allows tables which receive warnings during export to be included in the `.lst` file.

Examples

- To export all tables in the SAMPLE database (using default values for all options), issue:

```
db2move sample export
```

- To export all tables created by `userid1` or user IDs LIKE `us%rid2`, and with the name `tbname1` or table names LIKE `%tbname2`, issue:

```
db2move sample export -tc userid1,us*rid2 -tn tbname1,*tbname2
```

- To import all tables in the SAMPLE database (LOB paths `D:\LOBPATH1` and `C:\LOBPATH2` are to be searched for LOB files; this example is applicable to Windows operating systems only), issue:

```
db2move sample import -l D:\LOBPATH1,C:\LOBPATH2
```

- To load all tables in the SAMPLE database (`/home/userid/lobpath` subdirectory and the `tmp` subdirectory are to be searched for LOB files; this example is applicable to Linux and UNIX systems only), issue:

```
db2move sample load -l /home/userid/lobpath,/tmp
```

- To import all tables in the SAMPLE database in REPLACE mode using the specified user ID and password, issue:

```
db2move sample import -io replace -u userid -p password
```

- To duplicate schema `schema1` from source database `dbsrc` to target database `dbtgt`, issue:

```
db2move dbsrc COPY -sn schema1 -co TARGET_DB dbtgt USER myuser1 USING mypass1
```

- To duplicate schema `schema1` from source database `dbsrc` to target database `dbtgt`, rename the schema to `newschema1` on the target, and map source table space `ts1` to `ts2` on the target, issue:

```
db2move dbsrc COPY -sn schema1 -co TARGET_DB dbtgt USER myuser1 USING mypass1
     SCHEMA_MAP ((schema1,newschema1)) TABLESPACE_MAP ((ts1,ts2), SYS_ANY))
```

Usage notes

- A db2move EXPORT, followed by a db2move IMPORT/LOAD, facilitates the movement of table data. It is necessary to manually move all other database

objects associated with the tables (such as aliases, views, or triggers) as well as objects that these tables may depend on (such as user-defined types or user-defined functions).

- If the IMPORT action with the CREATE or REPLACE_CREATE option is used to create the tables on the target database (both options are deprecated and may be removed in a future release), then the limitations outlined in "Imported table re-creation" are imposed. If unexpected errors are encountered during the db2move import phase when the REPLACE_CREATE option is used, examine the appropriate `tabnnn.msg` message file and consider whether the errors might be the result of the limitations on table creation.
- Tables that contain GENERATED ALWAYS identity columns cannot be imported or loaded using db2move. You can, however, manually import or load these tables. For more information, see "Identity column load considerations" or "Identity column import considerations".
- When export, import, or load APIs are called by db2move, the **FileTypeMod** parameter is set to lobsinfile. That is, LOB data is kept in files that are separate from the PC/IXF file, for every table.
- The LOAD command must be run locally on the machine where the database and the data file reside.
- When using db2move LOAD and logretain is enabled for the database (the database is recoverable):
  - If the NONRECOVERABLE option is not specified, then db2move will invoke the db2Load API using the default COPY NO option, and the table spaces where the loaded tables reside are placed in the Backup Pending state upon completion of the utility (a full database or table space backup is required to take the table spaces out of the Backup Pending state).
  - If the NONRECOVERABLE option is specified, the table spaces are not placed in backup-pending state, however if rollforward recovery is performed later, the table is marked inaccessible and it must be dropped. For more information on Load recoverability options, see "Options for improving load performance".
- Performance for the db2move command with the IMPORT or LOAD actions can be improved by altering the default buffer pool, IBMDEFAULTBP, and by updating the configuration parameters **sortheap**, **util_heap_sz**, **logfilsiz**, and **logprimary**.

Files Required/Generated When Using EXPORT:

- Input: None.
- Output:

  EXPORT.out
  The summarized result of the EXPORT action.

  db2move.lst
  The list of original table names, their corresponding PC/IXF file names (tabnnn.ixf), and message file names (tabnnn.msg). This list, the exported PC/IXF files, and LOB files (tabnnnc.yyy) are used as input to the db2move IMPORT or LOAD action.

  tabnnn.ixf
  The exported PC/IXF file of a specific table.

  tabnnn.msg
  The export message file of the corresponding table.

tabnnnc.yyy
The exported LOB files of a specific table.

"nnn" is the table number. "c" is a letter of the alphabet. "yyy" is a number ranging from 001 to 999.

These files are created only if the table being exported contains LOB data. If created, these LOB files are placed in the "lobpath" directories. There are a total of 26,000 possible names for the LOB files.

system.msg
The message file containing system messages for creating or deleting file or directory commands. This is only used if the action is EXPORT, and a LOB path is specified.

Files Required/Generated When Using IMPORT:

- Input:

  db2move.lst
  An output file from the EXPORT action.

  tabnnn.ixf
  An output file from the EXPORT action.

  tabnnnc.yyy
  An output file from the EXPORT action.

- Output:

  IMPORT.out
  The summarized result of the IMPORT action.

  tabnnn.msg
  The import message file of the corresponding table.

Files Required/Generated When Using LOAD:

- Input:

  db2move.lst
  An output file from the EXPORT action.

  tabnnn.ixf
  An output file from the EXPORT action.

  tabnnnc.yyy
  An output file from the EXPORT action.

- Output:

  LOAD.out
  The summarized result of the LOAD action.

  tabnnn.msg
  The LOAD message file of the corresponding table.

Files Required/Generated When Using COPY:

- Input: None
- Output:

  COPYSCHEMA.msg
  An output file containing messages generated during the COPY operation.

COPYSCHEMA.err
An output file containing an error message for each error encountered during the COPY operation, including DDL statements for each object which could not be recreated on the target database.

LOADTABLE.msg
An output file containing messages generated by each invocation of the Load utility (used to repopulate data on the target database).

LOADTABLE.err
An output file containing the names of tables that either encountered a failure during Load or still need to be populated on the target database. See the "Restarting a failed copy schema operation" topic for more details.

These files are timestamped and all files that are generated from one run will have the same timestamp.

Performing a redirected restore using an automatically generated script

When you perform a redirected restore operation, you need to specify the locations of physical containers stored in the backup image and provide the complete set of containers for each table space that will be altered. Use the following procedure to generate a redirected restore script based on an existing backup image, modify the generated script, then run the script to perform the redirected restore.

You can perform a redirected restore only if the database has been previously backed up using the DB2 backup utility.

- If the database exists, you must be able to connect to it in order to generate the script. Therefore, if the database requires migration or crash recovery this must be done before you attempt to generate a redirected restore script.
- If you are working in a partitioned database environment, and the target database does not exist, you cannot run the command to generate the redirected restore script concurrently on all database partitions. Instead, the command to generate the redirected restore script must be run one database partition at a time, starting from the catalog partition.

  Alternatively, you can first create a dummy database with the same name as your target database. After the dummy database has been created, you can then generate the redirected restore script concurrently on all database partitions.
- Even if you specify the REPLACE EXISTING option when you issue the RESTORE command to generate the script, the REPLACE EXISTING option will appear in the script commented out.
- For security reasons, your password will not appear in the generated script. You need to fill in the password manually.
- You cannot generate a script for redirected restore using the Restore Wizard in the Control Center.

To perform a redirected restore using a script:

1. Use the restore utility to generate a redirected restore script. The restore utility can be invoked through the command line processor (CLP) or the db2Restore application programming interface (API). The following is an example of the RESTORE DATABASE command with the REDIRECT option and the GENERATE SCRIPT option:

```
db2 restore db test from /home/jseifert/backups taken at 20050304090733
      redirect generate script test_node0000.clp
```

This creates a redirected restore script on the client called `test_node0000.clp`.

2. Open the redirected restore script in a text editor to make any modifications that are required. You can modify:
   - Restore options
   - Automatic storage paths
   - Container layout and paths
3. Run the modified redirected restore script. For example:

```
db2 -tvf test_node0000.clp
```

RESTORE DATABASE

The RESTORE DATABASE command recreates a damaged or corrupted database that has been backed up using the DB2 backup utility. The restored database is in the same state that it was in when the backup copy was made. This utility can also overwrite a database with a different image or restore the backup copy to a new database.

For information on the restore operations supported by DB2 database systems between different operating systems and hardware platforms, see "Backup and restore operations between different operating systems and hardware platforms" in the *Data Recovery and High Availability Guide and Reference*.

The restore utility can also be used to restore backup images that were produced on DB2 Universal Database Version 8. If a migration is required, it will be invoked automatically at the end of the restore operation.

If, at the time of the backup operation, the database was enabled for rollforward recovery, the database can be brought to its previous state by invoking the rollforward utility after successful completion of a restore operation.

This utility can also restore a table space level backup.

Incremental images and images only capturing differences from the time of the previous capture (called a "delta image") cannot be restored when there is a difference in operating systems or word size (32-bit or 64-bit).

Following a successful restore operation from one environment to a different environment, no incremental or delta backups are allowed until a non-incremental backup is taken. (This is not a limitation following a restore operation within the same environment.)

Even with a successful restore operation from one environment to a different environment, there are some considerations: packages must be rebound before use (using the BIND command, the REBIND command, or the db2rbind utility); SQL procedures must be dropped and recreated; and all external libraries must be rebuilt on the new platform. (These are not considerations when restoring to the same environment.)

Scope

This command only affects the node on which it is executed.

Authorization

To restore to an existing database, one of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:

- *sysadm*
- *sysctrl*

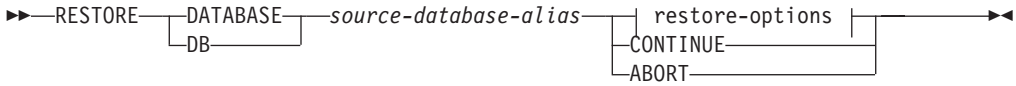
Required connection

The required connection will vary based on the type of restore action:

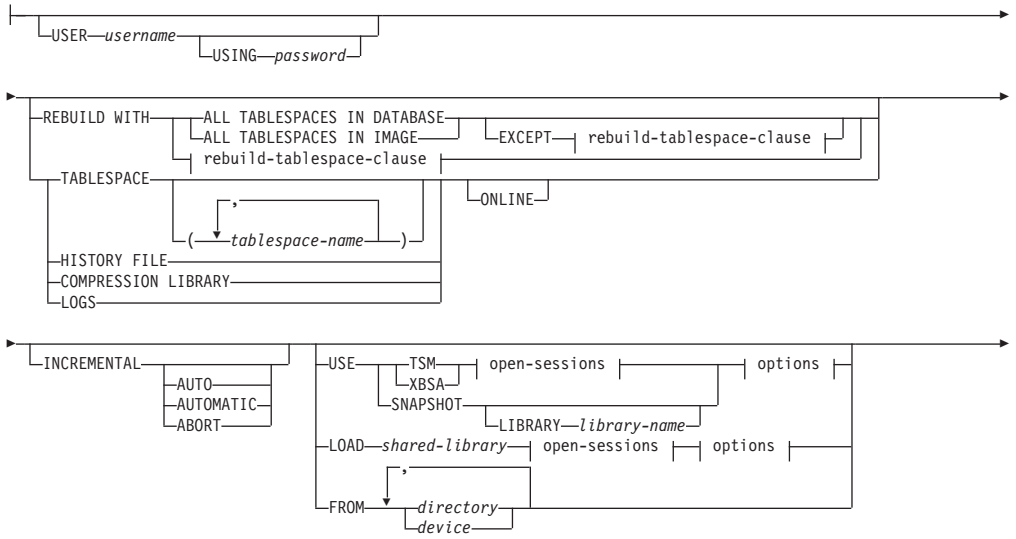
- You require a database connection, to restore to an existing database. This command automatically establishes an exclusive connection to the specified database.
- You require an instance and a database connection, to restore to a new database. The instance attachment is required to create the database.

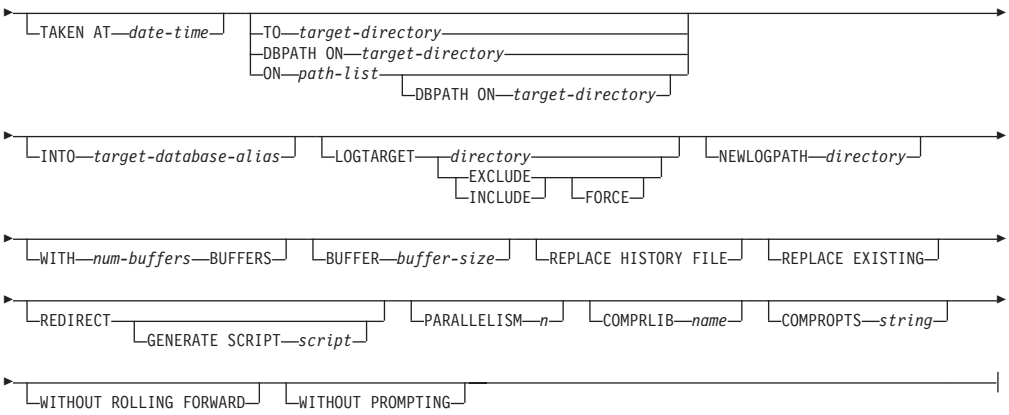
  To restore to a new database at an instance different from the current instance, it is necessary to first attach to the instance where the new database will reside. The new instance can be local or remote. The current instance is defined by the value of the DB2INSTANCE environment variable.
- For snapshot restore, *instance* and *database* connections are required.

Command syntax

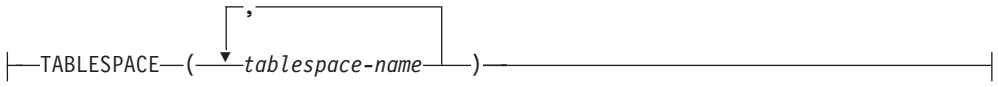


restore-options:

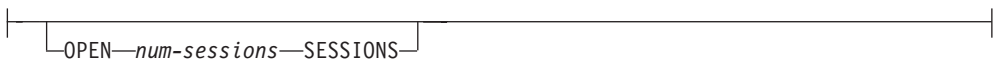




rebuild-tablespace-clause:



open-sessions:



options:

Command parameters

DATABASE *source-database-alias*
Alias of the source database from which the backup was taken.

CONTINUE
Specifies that the containers have been redefined, and that the final step in a redirected restore operation should be performed.

ABORT
This parameter:

- Stops a redirected restore operation. This is useful when an error has occurred that requires one or more steps to be repeated. After RESTORE DATABASE with the ABORT option has been issued, each step of a redirected restore operation must be repeated, including RESTORE DATABASE with the REDIRECT option.
- Terminates an incremental restore operation before completion.

USER *username*
Identifies the user name under which the database is to be restored.

USING *password*
The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

REBUILD WITH ALL TABLESPACES IN DATABASE
Restores the database with all the table spaces known to the database at the time of the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN DATABASE EXCEPT *rebuild-tablespace-clause*
Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE
Restores the database with only the table spaces in the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE EXCEPT *rebuild-tablespace-clause*
Restores the database with only the table spaces in the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH *rebuild-tablespace-clause*
Restores the database with only the list of table spaces specified. This restore overwrites a database if it already exists.

TABLESPACE *tablespace-name*
A list of names used to specify the table spaces that are to be restored.

ONLINE
This keyword, applicable only when performing a table space-level restore operation, is specified to allow a backup image to be restored online. This means that other agents can connect to the database while the backup image is being restored, and that the data in other table spaces will be available while the specified table spaces are being restored.

HISTORY FILE
This keyword is specified to restore only the history file from the backup image.

COMPRESSION LIBRARY
This keyword is specified to restore only the compression library from the backup image. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.

LOGS This keyword is specified to restore only the set of log files contained in the backup image. If the backup image does not contain any log files, the restore operation will fail. If this option is specified, the LOGTARGET option must also be specified.

INCREMENTAL
Without additional parameters, INCREMENTAL specifies a manual cumulative restore operation. During manual restore the user must issue each restore command manually for each image involved in the restore. Do so according to the following order: last, first, second, third and so on up to and including the last image.

INCREMENTAL AUTOMATIC/AUTO
Specifies an automatic cumulative restore operation.

INCREMENTAL ABORT
Specifies abortion of an in-progress manual cumulative restore operation.

USE

TSM Specifies that the database is to be restored from output managed by Tivoli Storage Manager.

XBSA Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

SNAPSHOT
Specifies that the data is to be restored from a snapshot backup.

You cannot use the SNAPSHOT parameter with any of the following parameters:

- INCREMENTAL
- TO
- ON
- DBPATH ON
- INTO
- NEWLOGPATH
- WITH *num-buffers* BUFFERS
- BUFFER
- REDIRECT
- REPLACE HISTORY FILE
- COMPRESSION LIBRARY
- PARALLELISM
- COMPRLIB
- OPEN *num-sessions* SESSIONS
- HISTORY FILE
- LOGS

Also, you cannot use the SNAPSHOT parameter with any restore operation that involves a table space list, which includes the REBUILD WITH option.

The default behavior when restoring data from a snapshot backup image will be a FULL DATABASE OFFLINE restore of all paths that make up the database including all containers, local volume directory, database path (DBPATH), primary log and mirror log paths of the most recent snapshot backup if no timestamp is provided (INCLUDE LOGS is the default for all snapshot backups unless EXCLUDE LOGS is explicitly stated). If a timestamp is provided, then that snapshot backup image will be restored.

LIBRARY *library-name*
Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage® SAN Volume Controller
- IBM Enterprise Storage Server® Model 800
- IBM System Storage™ DS6000™
- IBM System Storage DS8000™
- IBM System Storage N Series
- NetApp V-series

If you have other storage hardware, and a DB2 ACS API driver for that storage hardware, you can use the LIBRARY parameter to specify the DB2 ACS API driver.

The value of the LIBRARY parameter is a fully-qualified library file name.

OPTIONS

"options-string"
Specifies options to be used for the restore operation. The string will be passed to DB2 ACS API driver exactly as it was entered, without the double quotation marks. You cannot use the **VENDOROPT** database configuration parameter to specify vendor-specific options for snapshot restore operations. You must use the OPTIONS parameter of the restore utilities instead.

@file-name
Specifies that the options to be used for the restore operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library. The file must be a fully qualified file name.

OPEN *num-sessions* **SESSIONS**
Specifies the number of I/O sessions that are to be used with TSM or the vendor product.

FROM *directory/device*
The fully qualified path name of the directory or device on which the backup image resides. If USE TSM, FROM, and LOAD are omitted, the default value is the current working directory of the client machine. This target directory or device must exist on the target server/instance.

If several items are specified, and the last item is a tape device, the user is prompted for another tape. Valid response options are:

c Continue. Continue using the device that generated the warning message (for example, continue when a new tape has been mounted).

d Device terminate. Stop using *only* the device that generated the warning message (for example, terminate when there are no more tapes).

t Terminate. Abort the restore operation after the user has failed to perform some action requested by the utility.

LOAD *shared-library*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. The name can contain a full path. If the full path is not given, the value defaults to the path on which the user exit program resides.

TAKEN AT *date-time*
The time stamp of the database backup image. The time stamp is displayed after successful completion of a backup operation, and is part of the path name for the backup image. It is specified in the form *yyyymmddhhmmss*. A partial time stamp can also be specified. For example, if two different backup images with time stamps 20021001010101 and 20021002010101 exist, specifying `20021002` causes the image with time stamp `20021002010101` to be used. If a value for this parameter is not specified, there must be only one backup image on the source media.

TO *target-directory*

This parameter states the target database directory. This parameter is ignored if the utility is restoring to an existing database. The drive and directory that you specify must be local. If the backup image contains a database that is enabled for automatic storage then only the database directory changes, the storage paths associated with the database do not change.

DBPATH ON *target-directory*

This parameter states the target database directory. This parameter is ignored if the utility is restoring to an existing database. The drive and directory that you specify must be local. If the backup image contains a database that is enabled for automatic storage and the ON parameter is not specified then this parameter is synonymous with the TO parameter and only the database directory changes, the storage paths associated with the database do not change.

ON *path-list*

This parameter redefines the storage paths associated with an automatic storage database. Using this parameter with a database that is not enabled for automatic storage results in an error (SQL20321N). The existing storage paths as defined within the backup image are no longer used and automatic storage table spaces are automatically redirected to the new paths. If this parameter is not specified for an automatic storage database then the storage paths remain as they are defined within the backup image.

One or more paths can be specified, each separated by a comma. Each path must have an absolute path name and it must exist locally. If the database does not already exist on disk and the DBPATH ON parameter is not specified then the first path is used as the target database directory.

For a multi-partition database the ON *path-list* option can only be specified on the catalog partition. The catalog partition must be restored before any other partitions are restored when the ON option is used. The restore of the catalog-partition with new storage paths will place all non-catalog nodes in a RESTORE_PENDING state. The non-catalog nodes can then be restored in parallel without specifying the ON clause in the restore command.

In general, the same storage paths must be used for each partition in a multi-partition database and they must all exist prior to executing the RESTORE DATABASE command. One exception to this is where database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You use the argument `" $N" ([blank]$N)` to indicate a database partition expression. A database partition expression can be used anywhere in the storage path, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path after the database partition expression is evaluated. If there is no space character in the storage path after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms:

Table 49. . Operators are evaluated from left to right. % represents the modulus operator. The database partition number in the examples is assumed to be 10.

Syntax	Example	Value
[blank]$N	″ $N″	10
[blank]$N+[number]	″ $N+100″	110
[blank]$N%[number]	″ $N%5″	0
[blank]$N+[number]%[number]	″ $N+1%5″	1
[blank]$N%[number]+[number]	″ $N%4+2″	4
[a] % is modulus.		

INTO *target-database-alias*
The target database alias. If the target database does not exist, it is created.

When you restore a database backup to an existing database, the restored database inherits the alias and database name of the existing database. When you restore a database backup to a nonexistent database, the new database is created with the alias and database name that you specify. This new database name must be unique on the system where you restore it.

LOGTARGET *directory*
Non-snapshot restores:

The absolute path name of an existing directory on the database server, to be used as the target directory for extracting log files from a backup image. If this option is specified, any log files contained within the backup image will be extracted into the target directory. If this option is not specified, log files contained within a backup image will not be extracted. To extract only the log files from the backup image, specify the LOGS option.

Snapshot restores:

INCLUDE
Restore log directory volumes from the snapshot image. If this option is specified and the backup image contains log directories, then they will be restored. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If existing log directories on disk conflict with the log directories in the backup image, then an error will be returned.

EXCLUDE
Do not restore log directory volumes. If this option is specified, then no log directories will be restored from the backup image. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If a path belonging to the database is restored and a log directory will implicitly be restored because of this, thus causing a log directory to be overwritten, an error will be returned.

FORCE
Allow existing log directories in the current database to be overwritten and replaced when restoring the snapshot image. Without this option, existing log directories and log files on disk which conflict with log directories in the snapshot image will cause the restore to fail. Use this option to indicate that the restore can overwrite and replace those existing log directories.

Note: Use this option with caution, and always ensure that you have backed up and archived all logs that might be required for recovery.

Note: If LOGTARGET is not specified, then the default is LOGTARGET EXCLUDE.

NEWLOGPATH *directory*
The absolute pathname of a directory that will be used for active log files after the restore operation. This parameter has the same function as the **newlogpath** database configuration parameter, except that its effect is limited to the restore operation in which it is specified. The parameter can be used when the log path in the backup image is not suitable for use after the restore operation; for example, when the path is no longer valid, or is being used by a different database.

WITH *num-buffers* **BUFFERS**
The number of buffers to be used. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. A larger number of buffers can be used to improve performance when multiple sources are being read from, or if the value of PARALLELISM has been increased.

BUFFER *buffer-size*
The size, in pages, of the buffer used for the restore operation. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers are allocated to be of the smallest acceptable size.

REPLACE HISTORY FILE
Specifies that the restore operation should replace the history file on disk with the history file from the backup image.

REPLACE EXISTING
If a database with the same alias as the target database alias already exists, this parameter specifies that the restore utility is to replace the existing database with the restored database. This is useful for scripts that invoke the restore utility, because the command line processor will not prompt the user to verify deletion of an existing database. If the WITHOUT PROMPTING parameter is specified, it is not necessary to specify REPLACE EXISTING, but in this case, the operation will fail if events occur that normally require user intervention.

REDIRECT
Specifies a redirected restore operation. To complete a redirected restore operation, this command should be followed by one or more SET TABLESPACE CONTAINERS commands, and then by a RESTORE DATABASE command with the CONTINUE option. All commands associated with a single redirected restore operation must be invoked from the same window or CLP session. A redirected restore operation cannot be performed against a table space that has automatic storage enabled.

GENERATE SCRIPT *script*
Creates a redirect restore script with the specified file name. The script name can be relative or absolute and the script will be generated on the client side. If the file cannot be created on the client side, an error message

(SQL9304N) will be returned. If the file already exists, it will be overwritten. Please see the examples below for further usage information.

WITHOUT ROLLING FORWARD
Specifies that the database is not to be put in rollforward pending state after it has been successfully restored.

If, following a successful restore operation, the database is in rollforward pending state, the ROLLFORWARD command must be invoked before the database can be used again.

If this option is specified when restoring from an online backup image, error SQL2537N will be returned.

If backup image is of a recoverable database then WITHOUT ROLLING FORWARD cannot be specified with REBUILD option.

PARALLELISM *n*
Specifies the number of buffer manipulators that are to be created during the restore operation. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value.

COMPRLIB *name*
Indicates the name of the library to be used to perform the decompression (e.g., `db2compr.dll` for Windows; `libdb2compr.so` for Linux/UNIX systems). The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the restore operation will fail.

COMPROPTS *string*
Describes a block of binary data that is passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte reversal or code page conversion are handled by the decompression library. If the first character of the data block is "@", the remainder of the data is interpreted by the DB2 database system as the name of a file residing on the server. The DB2 database system will then replace the contents of *string* with the contents of this file and pass the new value to the initialization routine instead. The maximum length for the string is 1 024 bytes.

WITHOUT PROMPTING
Specifies that the restore operation is to run unattended. Actions that normally require user intervention will return an error message. When using a removable media device, such as tape or diskette, the user is prompted when the device ends, even if this option is specified.

Examples

1. In the following example, the database WSDB is defined on all 4 database partitions, numbered 0 through 3. The path /dev3/backup is accessible from all database partitions. The following offline backup images are available from /dev3/backup:

```
wsdb.0.db2inst1.NODE0000.CATN0000.20020331234149.001
wsdb.0.db2inst1.NODE0001.CATN0000.20020331234427.001
wsdb.0.db2inst1.NODE0002.CATN0000.20020331234828.001
wsdb.0.db2inst1.NODE0003.CATN0000.20020331235235.001
```

To restore the catalog partition first, then all other database partitions of the WSDB database from the /dev3/backup directory, issue the following commands from one of the database partitions:

```
db2_all '<<+0< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234149
  INTO wsdb REPLACE EXISTING'
db2_all '<<+1< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234427
  INTO wsdb REPLACE EXISTING'
db2_all '<<+2< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234828
  INTO wsdb REPLACE EXISTING'
db2_all '<<+3< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331235235
  INTO wsdb REPLACE EXISTING'
```

The db2_all utility issues the restore command to each specified database partition. When performing a restore using `db2_all`, you should always specify REPLACE EXISTING and/or WITHOUT PROMPTING. Otherwise, if there is prompting, the operation will look like it is hanging. This is because db2_all does not support user prompting.

2. Following is a typical redirected restore scenario for a database whose alias is MYDB:
   a. Issue a RESTORE DATABASE command with the REDIRECT option.

      ```
      restore db mydb replace existing redirect
      ```

      After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

      ```
      restore db mydb abort
      ```

   b. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example:

      ```
      set tablespace containers for 5 using
         (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
      ```

      To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

   c. After successful completion of steps 1 and 2, issue:

      ```
      restore db mydb continue
      ```

      This is the final step of the redirected restore operation.

   d. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.
3. Following is a sample weekly incremental backup strategy for a recoverable database. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

   ```
   (Sun) backup db mydb use tsm
   (Mon) backup db mydb online incremental delta use tsm
   (Tue) backup db mydb online incremental delta use tsm
   (Wed) backup db mydb online incremental use tsm
   (Thu) backup db mydb online incremental delta use tsm
   (Fri) backup db mydb online incremental delta use tsm
   (Sat) backup db mydb online incremental use tsm
   ```

   For an automatic database restore of the images created on Friday morning, issue:

```
restore db mydb incremental automatic taken at (Fri)
```

For a manual database restore of the images created on Friday morning, issue:

```
restore db mydb incremental taken at (Fri)
restore db mydb incremental taken at (Sun)
restore db mydb incremental taken at (Wed)
restore db mydb incremental taken at (Thu)
restore db mydb incremental taken at (Fri)
```

4. To produce a backup image, which includes logs, for transportation to a remote site:

   ```
   backup db sample online to /dev3/backup include logs
   ```

   To restore that backup image, supply a LOGTARGET path and specify this path during ROLLFORWARD:

   ```
   restore db sample from /dev3/backup logtarget /dev3/logs
   rollforward db sample to end of logs and stop overflow log path /dev3/logs
   ```

5. To retrieve only the log files from a backup image that includes logs:

   ```
   restore db sample logs from /dev3/backup logtarget /dev3/logs
   ```

6. The USE TSM OPTIONS keywords can be used to specify the TSM information to use for the restore operation. On Windows platforms, omit the -fromowner option.

   - Specifying a delimited string:

     ```
     restore db sample use TSM options '"-fromnode=bar -fromowner=dmcinnis"'
     ```

   - Specifying a fully qualified file:

     ```
     restore db sample use TSM options @/u/dmcinnis/myoptions.txt
     ```

     The file myoptions.txt contains the following information: -fromnode=bar -fromowner=dmcinnis

7. The following is a simple restore of a multi-partition automatic storage enabled database with new storage paths. The database was originally created with one storage path, `/myPath0:`

   - On the catalog partition issue: `restore db mydb on /myPath1,/myPath2`
   - On all non-catalog partitions issue: `restore db mydb`

8. A script output of the following command on a non-auto storage database:

   ```
   restore db sample from /home/jseifert/backups taken at 20050301100417 redirect
   generate script SAMPLE_NODE0000.clp
   ```

   would look like this:

   ```
   -- ******************************************************************************
   -- ** automatically created redirect restore script
   -- ******************************************************************************
   UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
   SET CLIENT ATTACH_DBPARTITIONNUM  0;
   SET CLIENT CONNECT_DBPARTITIONNUM 0;
   -- ******************************************************************************
   -- ** initialize redirected restore
   -- ******************************************************************************
   RESTORE DATABASE SAMPLE
   -- USER  '<username>'
   -- USING '<password>'
   FROM '/home/jseifert/backups'
   TAKEN AT 20050301100417
   -- DBPATH ON '<target-directory>'
   INTO SAMPLE
   -- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/SQLOGDIR/'
   -- WITH <num-buff> BUFFERS
   ```

```
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- ******************************************************************************
-- ** tablespace definition
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = SYSCATSPACE
-- **   Tablespace ID                         = 0
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 5572
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = TEMPSPACE1
-- **   Tablespace ID                         = 1
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = System Temporary data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 0
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = USERSPACE1
-- **   Tablespace ID                         = 2
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Total number of pages                 = 1
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0002.0'
);
-- ******************************************************************************
-- ** Tablespace name                         = DMS
-- **   Tablespace ID                         = 3
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
```

```
-- **   High water mark (pages)                 = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   /tmp/dms1                                                      1000
, FILE   /tmp/dms2                                                      1000
);
-- ******************************************************************************
-- ** Tablespace name                         = RAW
-- **   Tablespace ID                         = 4
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
-- **   High water mark (pages)               = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                            1000
, DEVICE '/dev/hdb2'                                            1000
);
-- ******************************************************************************
-- ** start redirect restore
-- ******************************************************************************
RESTORE DATABASE SAMPLE CONTINUE;
-- ******************************************************************************
-- ** end of file
-- ******************************************************************************
```

9. A script output of the following command on an automatic storage database:

```
restore db test from /home/jseifert/backups taken at 20050304090733 redirect
generate script TEST_NODE0000.clp
```

would look like this:

```
-- ******************************************************************************
-- ** automatically created redirect restore script
-- ******************************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON TEST_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM  0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
-- ******************************************************************************
-- ** initialize redirected restore
-- ******************************************************************************
RESTORE DATABASE TEST
-- USER  '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050304090733
ON '/home/jseifert'
-- DBPATH ON <target-directory>
INTO TEST
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00002/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
```

```
;
-- ******************************************************************************
-- ** tablespace definition
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = SYSCATSPACE
-- **   Tablespace ID                         = 0
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 4
-- **   Using automatic storage               = Yes
-- **   Auto-resize enabled                   = Yes
-- **   Total number of pages                 = 6144
-- **   Number of usable pages                = 6140
-- **   High water mark (pages)               = 5968
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = TEMPSPACE1
-- **   Tablespace ID                         = 1
-- **   Tablespace Type                       = System managed space
-- **   Tablespace Content Type               = System Temporary data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = Yes
-- **   Total number of pages                 = 0
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = USERSPACE1
-- **   Tablespace ID                         = 2
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = Yes
-- **   Auto-resize enabled                   = Yes
-- **   Total number of pages                 = 256
-- **   Number of usable pages                = 224
-- **   High water mark (pages)               = 96
-- ******************************************************************************
-- ******************************************************************************
-- ** Tablespace name                         = DMS
-- **   Tablespace ID                         = 3
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
-- **   Auto-resize enabled                   = No
-- **   Total number of pages                 = 2000
-- **   Number of usable pages                = 1960
-- **   High water mark (pages)               = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                        1000
, FILE   '/tmp/dms2'                                        1000
);
-- ******************************************************************************
-- ** Tablespace name                         = RAW
-- **   Tablespace ID                         = 4
-- **   Tablespace Type                       = Database managed space
-- **   Tablespace Content Type               = Any data
-- **   Tablespace Page size (bytes)          = 4096
-- **   Tablespace Extent size (pages)        = 32
-- **   Using automatic storage               = No
```

```
-- **   Auto-resize enabled                     = No
-- **   Total number of pages                   = 2000
-- **   Number of usable pages                  = 1960
-- **   High water mark (pages)                 = 96
-- ******************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                          1000
, DEVICE '/dev/hdb2'                                          1000
);
-- ******************************************************************************
-- ** start redirect restore
-- ******************************************************************************
RESTORE DATABASE TEST CONTINUE;
-- ******************************************************************************
-- ** end of file
-- ******************************************************************************
```

10. The following are examples of the RESTORE DB command using the SNAPSHOT option:

    Restore log directory volumes from the snapshot image and do not prompt.

    ```
    db2 restore db sample use snapshot LOGTARGET INCLUDE without prompting
    ```

    Do not restore log directory volumes and do not prompt.

    ```
    db2 restore db sample use snapshot LOGTARGET EXCLUDE without prompting
    ```

    Do not restore log directory volumes and do not prompt. When LOGTARGET is not specified, then the default is LOGTARGET EXCLUDE.

    ```
    db2 restore db sample use snapshot without prompting
    ```

    Allow existing log directories in the current database to be overwritten and replaced when restoring the snapshot image containing conflicting log directories, without prompting.

    ```
    db2 restore db sample use snapshot LOGTARGET EXCLUDE FORCE without prompting
    ```

    Allow existing log directories in the current database to be overwritten and replaced when restoring the snapshot image containing conflicting log directories, without prompting.

    ```
    db2 restore db sample use snapshot LOGTARGET INCLUDE FORCE without prompting
    ```

Usage notes

- A RESTORE DATABASE command of the form `db2 restore db <name>` will perform a full database restore with a database image and will perform a table space restore operation of the table spaces found in a table space image. A RESTORE DATABASE command of the form `db2 restore db <name> tablespace` performs a table space restore of the table spaces found in the image. In addition, if a list of table spaces is provided with such a command, the explicitly listed table spaces are restored.
- Following the restore operation of an online backup, you must perform a roll-forward recovery.
- If a backup image is compressed, the DB2 database system detects this and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. Otherwise, a check is made to see if a library is stored in the backup image and if the library exists, it is used. Finally, if there is not library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is to be restored from a backup image (either explicitly by specifying the COMPRESSION LIBRARY option or implicitly by performing a normal restore of a compressed backup), the restore operation must be done on the same platform and operating system that the backup was

taken on. If the platform the backup was taken on is not the same as the platform that the restore is being done on, the restore operation will fail, even if DB2 normally supports cross-platform restores involving the two systems.

- To restore log files from the backup image that contains them, the LOGTARGET option must be specified, providing the fully qualified and valid path that exists on the DB2 server. If those conditions are satisfied, the restore utility will write the log files from the image to the target path. If a LOGTARGET is specified during a restore of a backup image that does not include logs, the restore operation will return an error before attempting to restore any table space data. A restore operation will also fail with an error if an invalid, or read-only, LOGTARGET path is specified.
- If any log files exist in the LOGTARGET path at the time the RESTORE DATABASE command is issued, a warning prompt will be returned to the user. This warning will not be returned if WITHOUT PROMPTING is specified.
- During a restore operation where a LOGTARGET is specified, if any log file cannot be extracted, the restore operation will fail and return an error. If any of the log files being extracted from the backup image have the same name as an existing file in the LOGTARGET path, the restore operation will fail and an error will be returned. The restore database utility will not overwrite existing log files in the LOGTARGET directory.
- You can also restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the LOGTARGET path. Specifying the LOGS option without a LOGTARGET path will result in an error. If any problem occurs while restoring log files in this mode of operation, the restore operation will terminate immediately and an error will be returned.
- During an automatic incremental restore operation, only the log files included in the target image of the restore operation will be retrieved from the backup image. Any log files included in intermediate images referenced during the incremental restore process will not be extracted from those intermediate backup images. During a manual incremental restore operation, the LOGTARGET path should only be specified with the final restore command to be issued.
- Offline full database backups as well as offline incremental database backups can be restored to a later database version, whereas online backups cannot. For multi-partition databases, the catalog partition must first be restored individually, followed by the remaining database partitions (in parallel or serial). However, the implicit database migration done by the restore operation can fail. In a multi-partition database it can fail on one or more database partitions. In this case, you can follow the RESTORE DATABASE command with a single MIGRATE DATABASE command issued from the catalog partition to migrate the database successfully.

Snapshot restore

Like a traditional (non-snapshot) restore, the default behavior when restoring a snapshot backup image will be to NOT restore the log directories —LOGTARGET EXCLUDE.

If the DB2 manager detects that any log directory's group ID is shared among any of the other paths to be restored, then an error is returned. In this case, LOGTARGET INCLUDE or LOGTARGET INCLUDE FORCE must be specified, as the log directories must be part of the restore.

The DB2 manager will make all efforts to save existing log directories (primary, mirror and overflow) before the restore of the paths from the backup image takes place.

If you wish the log directories to be restored and the DB2 manager detects that the pre-existing log directories on disk conflict with the log directories in the backup image, then the DB2 manager will report an error. In such a case, if you have specified LOGTARGET INCLUDE FORCE, then this error will be suppressed and the log directories from the image will be restored, deleting whatever existed beforehand.

There is a special case in which the LOGTARGET EXCLUDE option is specified and a log directory path resides under the database directory (i.e., /NODExxxx/SQLxxxxx/SQLOGDIR/). In this case, a restore would still overwrite the log directory as the database path, and all of the contents beneath it, would be restored. If the DB2 manager detects this scenario and log files exist in this log directory, then an error will be reported. If you specify LOGTARGET EXCLUDE FORCE, then this error will be suppressed and those log directories from the backup image will overwrite the conflicting log directories on disk.

High availability through suspended I/O and online split mirror support

IBM Data Server suspended I/O support enables you to split mirrored copies of your primary database without taking the database off-line. You can use this to very quickly create a standby database to take over if the primary database fails.

Disk mirroring is the process of writing data to two separate hard disks at the same time. One copy of the data is called a mirror of the other. Splitting a mirror is the process of separating the two copies.

You can use disk mirroring to maintain a secondary copy of your primary database. You can use IBM Data Server suspended I/O functionality to split the primary and secondary mirrored copies of the database without taking the database off-line. Once the primary and secondary databases copies are split, the secondary database can take over operations if the primary database fails.

If you would rather not back up a large database using the IBM Data Server backup utility, you can make copies from a mirrored image by using suspended I/O and the split mirror function. This approach also:

- Eliminates backup operation overhead from the production machine
- Represents a fast way to clone systems
- Represents a fast implementation of idle standby failover. There is no initial restore operation, and if a rollforward operation proves to be too slow, or encounters errors, reinitialization is very fast.

The db2inidb command initializes the split mirror so that it can be used:

- As a clone database
- As a standby database
- As a backup image

This command can only be issued against a split mirror, and it must be run before the split mirror can be used.

In a partitioned database environment, you do not have to suspend I/O writes on all database partitions simultaneously. You can suspend a subset of one or more

database partitions to create split mirrors for performing offline backups. If the catalog partition is included in the subset, it must be the last database partition to be suspended.

In a partitioned database environment, the db2inidb command must be run on every database partition before the split image from any of the database partitions can be used. The tool can be run on all database partitions simultaneously using the db2_all command. If; however, you are using the RELOCATE USING option, you cannot use the db2_all command to run db2inidb on all of the database partitions simultaneously. A separate configuration file must be supplied for each database partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being changed, every database partition will be affected and the db2relocatedb command must be run with a separate configuration file on each database partition. If containers belonging to a single database partition are being moved, the db2relocatedb command only needs to be run once on that database partition.

Note: Ensure that the split mirror contains all containers and directories which comprise the database, including the volume directory. To gather this information, refer to the DBPATHS administrative view, which shows all the files and directories of the database that need to be split.

db2inidb - Initialize a mirrored database

Initializes a mirrored database in a split mirror environment. The mirrored database can be initialized as a clone of the primary database, placed in roll forward pending state, or used as a backup image to restore the primary database. This command can only be run against a split mirror database, and it must be run before the split mirror can be used.

Authorization

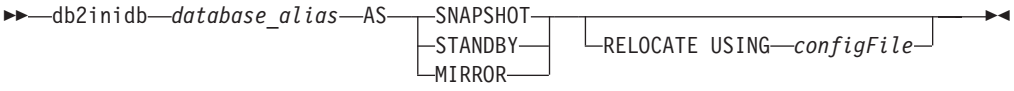
One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection

None

Command syntax



Command parameters

database_alias
Specifies the alias of the database to be initialized.

SNAPSHOT
Specifies that the mirrored database will be initialized as a clone of the primary database.

STANDBY
Specifies that the database will be placed in roll forward pending state. New logs from the primary database can be fetched and applied to the standby database. The standby database can then be used in place of the primary database if it goes down.

MIRROR
Specifies that the mirrored database is to be used as a backup image which can be used to restore the primary database.

RELOCATE USING *configFile*
Specifies that the database files are to be relocated based on the information listed in the specified *configFile* prior to initializing the database as a snapshot, standby, or mirror. The format of *configFile* is described in "db2relocatedb - Relocate database" on page 358.

Usage notes

Do not issue the `db2 connect to` *database-alias* operation before issuing the db2inidb *database_alias* as mirror command. Attempting to connect to a split mirror database before initializing it erases the log files needed during roll forward recovery. The connect sets your database back to the state it was in when you suspended the database. If the database is marked as consistent when it was suspended, the DB2 database system concludes there is no need for crash recovery and empties the logs for future use. If the logs have been emptied, attempting to roll forward results in the SQL4970N error message being returned.

In a partitioned database environment, db2inidb must be run on every database partition before the split mirror from any of the database partitions can be used. db2inidb can be run on all database partitions simultaneously using the db2_all command.

If, however, you are using the RELOCATE USING option, you cannot use the db2_all command to run db2inidb on all of the partitions simultaneously. A separate configuration file must be supplied for each partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being changed, every database partition will be affected and the db2relocatedb command must be run with a separate configuration file on each database partition. If containers belonging to a single database partition are being moved, the db2relocatedb command only needs to be run once on that database partition.

If the RELOCATE USING *configFile* parameter is specified and the database is relocated successfully, the specified *configFile* will be copied into the database directory and renamed to `db2path.cfg`. During a subsequent crash recovery or rollforward recovery, this file will be used to rename container paths as log files are being processed.

If a clone database is being initialized, the specified *configFile* will be automatically removed from the database directory after a crash recovery is completed.

If a standby database or mirrored database is being initialized, the specified *configFile* will be automatically removed from the database directory after a rollforward recovery is completed or canceled. New container paths can be added to the `db2path.cfg` file after db2inidb has been run. This would be necessary when CREATE or ALTER TABLESPACE operations are done on the original database and different paths must be used on the standby database.

db2relocatedb - Relocate database

This command renames a database, or relocates a database or part of a database (for example, the container and the log directory) as specified in the configuration file provided by the user. This tool makes the necessary changes to the DB2 instance and database support files.

Authorization

None

Command syntax

Command parameters

-f *configFilename*

Specifies the name of the file containing the configuration information necessary for relocating the database. This can be a relative or absolute file name. The format of the configuration file is:

```
DB_NAME=oldName,newName
DB_PATH=oldPath,newPath
INSTANCE=oldInst,newInst
NODENUM=nodeNumber
LOG_DIR=oldDirPath,newDirPath
CONT_PATH=oldContPath1,newContPath1
CONT_PATH=oldContPath2,newContPath2
...
STORAGE_PATH=oldStoragePath1,newStoragePath1
STORAGE_PATH=oldStoragePath2,newStoragePath2
...
```

Where:

DB_NAME

Specifies the name of the database being relocated. If the database name is being changed, both the old name and the new name must be specified. This is a required field.

DB_PATH

Specifies the original path of the database being relocated. If the database path is changing, both the old path and new path must be specified. This is a required field.

INSTANCE

Specifies the instance where the database exists. If the database is being moved to a new instance, both the old instance and new instance must be specified. This is a required field.

NODENUM

Specifies the node number for the database node being changed. The default is 0.

LOG_DIR

Specifies a change in the location of the log path. If the log path is being changed, both the old path and new path must be specified. This specification is optional if the log path resides under the database path, in which case the path is updated automatically.

CONT_PATH
Specifies a change in the location of table space containers. Both the old and new container path must be specified. Multiple CONT_PATH lines can be provided if there are multiple container path changes to be made. This specification is optional if the container paths reside under the database path, in which case the paths are updated automatically. If you are making changes to more than one container where the same old path is being replaced by a common new path, a single CONT_PATH entry can be used. In such a case, an asterisk (*) could be used both in the old and new paths as a wildcard.

STORAGE_PATH
This is only applicable to databases with automatic storage enabled. It specifies a change in the location of one of the storage paths for the database. Both the old storage path and the new storage path must be specified. Multiple STORAGE_PATH lines can be given if there are several storage path changes to be made.

Blank lines or lines beginning with a comment character (#) are ignored.

Examples

Example 1

To change the name of the database TESTDB to PRODDB in the instance db2inst1 that resides on the path /home/db2inst1, create the following configuration file:

```
DB_NAME=TESTDB,PRODDB
DB_PATH=/home/db2inst1
INSTANCE=db2inst1
NODENUM=0
```

Save the configuration file as relocate.cfg and use the following command to make the changes to the database files:

```
db2relocatedb -f relocate.cfg
```

Example 2

To move the database DATAB1 from the instance jsmith on the path /dbpath to the instance prodinst do the following:

1. Move the files in the directory /dbpath/jsmith to /dbpath/prodinst.
2. Use the following configuration file with the db2relocatedb command to make the changes to the database files:

```
DB_NAME=DATAB1
DB_PATH=/dbpath
INSTANCE=jsmith,prodinst
NODENUM=0
```

Example 3

The database PRODDB exists in the instance inst1 on the path /databases/PRODDB. The location of two table space containers needs to be changed as follows:

- SMS container /data/SMS1 needs to be moved to /DATA/NewSMS1.
- DMS container /data/DMS1 needs to be moved to /DATA/DMS1.

After the physical directories and files have been moved to the new locations, the following configuration file can be used with the db2relocatedb command to make changes to the database files so that they recognize the new locations:

```
DB_NAME=PRODDB
DB_PATH=/databases/PRODDB
INSTANCE=inst1
NODENUM=0
CONT_PATH=/data/SMS1,/DATA/NewSMS1
CONT_PATH=/data/DMS1,/DATA/DMS1
```

Example 4

The database TESTDB exists in the instance db2inst1 and was created on the path /databases/TESTDB. Table spaces were then created with the following containers:

```
TS1
TS2_Cont0
TS2_Cont1
/databases/TESTDB/TS3_Cont0
/databases/TESTDB/TS4/Cont0
/Data/TS5_Cont0
/dev/rTS5_Cont1
```

TESTDB is to be moved to a new system. The instance on the new system will be newinst and the location of the database will be /DB2.

When moving the database, all of the files that exist in the /databases/TESTDB/db2inst1 directory must be moved to the /DB2/newinst directory. This means that the first 5 containers will be relocated as part of this move. (The first 3 are relative to the database directory and the next 2 are relative to the database path.) Since these containers are located within the database directory or database path, they do not need to be listed in the configuration file. If the 2 remaining containers are to be moved to different locations on the new system, they must be listed in the configuration file.

After the physical directories and files have been moved to their new locations, the following configuration file can be used with db2relocatedb to make changes to the database files so that they recognize the new locations:

```
DB_NAME=TESTDB
DB_PATH=/databases/TESTDB,/DB2
INSTANCE=db2inst1,newinst
NODENUM=0
CONT_PATH=/Data/TS5_Cont0,/DB2/TESTDB/TS5_Cont0
CONT_PATH=/dev/rTS5_Cont1,/dev/rTESTDB_TS5_Cont1
```

Example 5

The database TESTDB has two database partitions on database partition servers 10 and 20. The instance is servinst and the database path is /home/servinst on both database partition servers. The name of the database is being changed to SERVDB and the database path is being changed to /databases on both database partition servers. In addition, the log directory is being changed on database partition server 20 from /testdb_logdir to /servdb_logdir.

Since changes are being made to both database partitions, a configuration file must be created for each database partition and db2relocatedb must be run on each database partition server with the corresponding configuration file.

On database partition server 10, the following configuration file will be used:

```
DB_NAME=TESTDB,SERVDB
DB_PATH=/home/servinst,/databases
INSTANCE=servinst
NODE_NUM=10
```

On database partition server 20, the following configuration file will be used:

```
DB_NAME=TESTDB,SERVDB
DB_PATH=/home/servinst,/databases
INSTANCE=servinst
NODE_NUM=20
LOG_DIR=/testdb_logdir,/servdb_logdir
```

Example 6

The database MAINDB exists in the instance `maininst` on the path `/home/maininst`. The location of four table space containers needs to be changed as follows:

```
/maininst_files/allconts/C0 needs to be moved to /MAINDB/C0
/maininst_files/allconts/C1 needs to be moved to /MAINDB/C1
/maininst_files/allconts/C2 needs to be moved to /MAINDB/C2
/maininst_files/allconts/C3 needs to be moved to /MAINDB/C3
```

After the physical directories and files are moved to the new locations, the following configuration file can be used with the db2relocatedb command to make changes to the database files so that they recognize the new locations.

A similar change is being made to all of the containers; that is, `/maininst_files/allconts/` is being replaced by `/MAINDB/` so that a single entry with the wildcard character can be used:

```
DB_NAME=MAINDB
DB_PATH=/home/maininst
INSTANCE=maininst
NODE_NUM=0
CONT_PATH=/maininst_files/allconts/*, /MAINDB/*
```

Usage notes

If the instance that a database belongs to is changing, the following must be done before running this command to ensure that changes to the instance and database support files are made:

- If a database is being moved to another instance, create the new instance.
- Copy the files and devices belonging to the databases being copied onto the system where the new instance resides. The path names must be changed as necessary. However, if there are already databases in the directory where the database files are moved to, you can mistakenly overwrite the existing sqldbdir file, thereby removing the references to the existing databases. In this scenario, the db2relocatedb utility cannot be used. Instead of db2relocatedb, an alternative is a redirected restore operation.
- Change the permission of the files/devices that were copied so that they are owned by the instance owner.

If the instance is changing, the tool must be run by the new instance owner.

In a partitioned database environment, this tool must be run against every database partition that requires changes. A separate configuration file must be supplied for each database partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being

changed, every database partition will be affected and the db2relocatedb command must be run with a separate configuration file on each database partition. If containers belonging to a single database partition are being moved, the db2relocatedb command only needs to be run once on that database partition.

You cannot use the db2relocatedb command to relocate a database that has a load in progress or is waiting for the completion of a LOAD RESTART or LOAD TERMINATE command.

Limitation: In a partitioned database environment, you cannot relocate an entire node if that node is one of two or more logical partitions that reside on the same device.

db2look - DB2 statistics and DDL extraction tool

Extracts the required Data Definition Language (DDL) statements to reproduce the database objects of a production database on a test database. The db2look command generates the DDL statements by object type.

This tool can generate the required UPDATE statements used to replicate the statistics on the objects in a test database. It can also be used to generate the UPDATE DATABASE CONFIGURATION and UPDATE DATABASE MANAGER CONFIGURATION commands and the db2set commands so that query optimizer-related configuration parameters and registry variables on the test database match those of the production database.

It is often advantageous to have a test system contain a subset of the production system's data. However, access plans selected for such a test system are not necessarily the same as those that would be selected for the production system. Both the catalog statistics and the configuration parameters for the test system must be updated to match those of the production system. Using this tool makes it possible to create a test database where access plans are similar to those that would be used on the production system.

You should check the DDL statements generated by the db2look command since they might not exactly reproduce all characteristics of the original SQL objects. For table spaces on partitioned database environments, DDL might not be complete if some database partitions are not active. Make sure all database partitions are active using the ACTIVATE command.

Authorization

SELECT privilege on the system catalog tables.

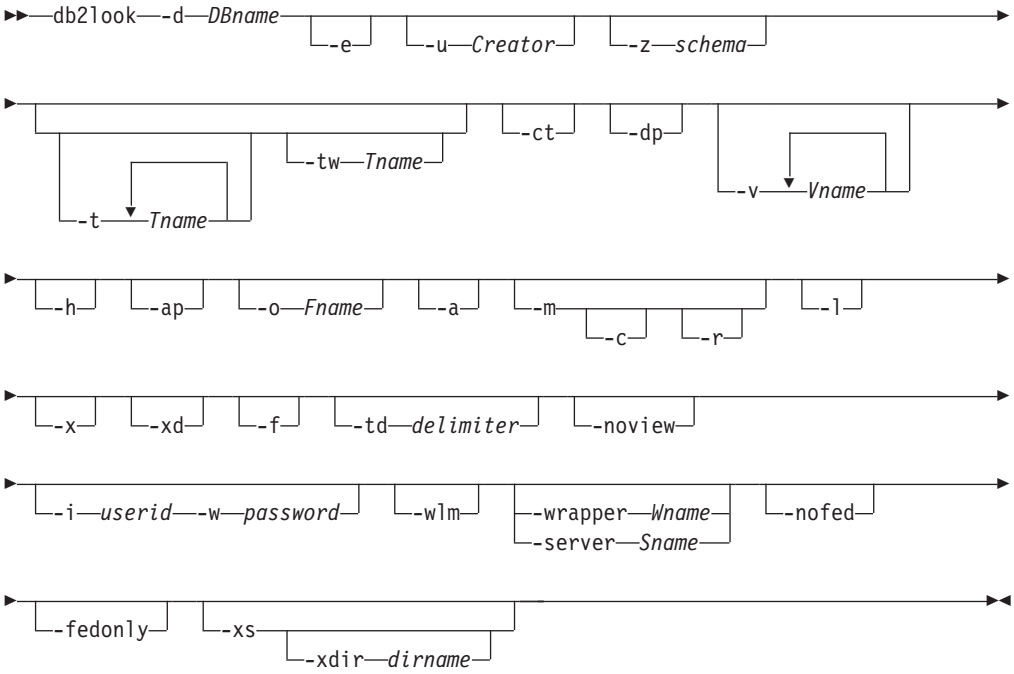
In some cases, such as generating table space container DDL (which calls the APIs sqlbotcq, sqlbftcq, and sqlbctcq), you will require one of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection

None

Command syntax



Command parameters

-d *DBname*

Alias name of the production database that is to be queried. *DBname* can be the name of a DB2 Database for Linux, UNIX, and Windows or DB2 Version 9.1 for z/OS (DB2 for z/OS) database. If the *DBname* is a DB2 for z/OS database, the db2look utility will extract the DDL and UPDATE statistics statements for OS/390 and z/OS objects. These DDL and UPDATE statistics statements are statements applicable to a DB2 Database for Linux, UNIX, and Windows database and not to a DB2 for z/OS database. This is useful for users who want to extract OS/390 and z/OS objects and recreate them in a DB2 Database for Linux, UNIX, and Windows database.

If *DBname* is a DB2 for z/OS database, the output of the db2look command is limited to the following:

- Generate DDL for tables, indexes, views, and user-defined distinct types
- Generate UPDATE statistics statements for tables, columns, column distributions and indexes

-e Extract DDL statements for database objects. DDL for the following database objects are extracted when using the -e option:

- Audit policies
- Schemas
- Tables
- Views
- Materialized query tables (MQT)
- Aliases
- Indexes
- Triggers

- Sequences
- User-defined distinct types
- Primary key, referential integrity, and check constraints
- User-defined structured types
- User-defined functions
- User-defined methods
- User-defined transforms
- Wrappers
- Servers
- User mappings
- Nicknames
- Type mappings
- Function templates
- Function mappings
- Index specifications
- Stored procedures
- Roles
- Trusted contexts
- Global variables
- Security label components
- Security policies
- Security labels

The DDL generated by the db2look command can be used to recreate user-defined functions successfully. However, the user source code that a particular user-defined function references (the EXTERNAL NAME clause, for example) must be available in order for the user-defined function to be usable.

-u *Creator*
Creator ID. Limits output to objects with this creator ID. If option -a is specified, this parameter is ignored. The output will not include any inoperative objects. To display inoperative objects, use the -a option.

-z *schema*
Schema name. Limits output to objects with this schema name. The output will not include any inoperative objects. To display inoperative objects, use the -a option. If this parameter is not specified, objects with all schema names are extracted. If the -a option is specified, this parameter is ignored. This option is ignored for the federated DDL.

-t *Tname1 Tname2 ... TnameN*
Table name list. Limits the output to particular tables in the table list. The maximum number of tables is 30. Table names are separated by a blank space. Case-sensitive names and double-byte character set (DBCS) names must be enclosed inside a backward slash and double quotation delimiter, for example, \" MyTabLe \". For multiple-word table names, the delimiters must be placed within quotation marks (for example, "\"My Table\"") to prevent the pairing from being evaluated word-by-word by the command line processor. If a multiple-word table name is not enclosed by the backward slash and double delimiter (for example, "My Table"), all words will be converted into uppercase and the db2look command will look for

an uppercase table (for example, ″MY TABLE″). When -t is used with -l, the combination does support partitioned tables in DB2 Version 9.5.

-tw *Tname*
Generates DDL for table names that match the pattern criteria specified by *Tname*. Also generates the DDL for all dependent objects of all returned tables. *Tname* can be a single value only. The underscore character (_) in *Tname* represents any single character. The percent sign (%) represents a string of zero or more characters. Any other character in *Tname* only represents itself. When -tw is specified, the -t option is ignored.

-ct Generate DDL by object creation time. Generating DDL by object creation time will not guarantee that all the object DDLs will be displayed in correct dependency order. The db2look command only supports the following options if the -ct option is also specified: -e, -a, -u, -z, -t, -tw, -v, -l, -noview, -wlm.

-dp Generate DROP statement before CREATE statement. The DROP statement might not work if there is an object that depends on the dropped object. For example, dropping a schema will fail if there is a table that depends on the dropped schema, or dropping a user-defined type/function will fail if there is any other type, function, trigger, or table that depends on it. For typed tables, the DROP TABLE HIERARCHY statement will be generated for the root table only. A DROP statement is not generated for index, primary and foreign keys, and constraints, because they are always dropped when the table is dropped. When a table has the RESTRICT ON DROP attribute, it cannot be dropped.

-v *Vname1 Vname2 ... VnameN*
Generates DDL for the specified views. The maximum number of views is 30. If the -t option is specified, the -v option is ignored. The rules governing case-sensitive, DBCS, and multiple-word table names also apply to view names.

-h Display help information. When this option is specified, all other options are ignored, and only the help information is displayed.

-ap Generates the required AUDIT USING statements to associate audit policies with other database objects.

-o *Fname*
Write the output to *filename*.`sql`. If this option is not specified, output is written to standard output. If a filename is specified with an extension, the output will be written into that file.

-a When this option is specified the output is not limited to the objects created under a particular creator ID. All objects, including inoperative objects, created by all users are considered. For example, if this option is specified with the -e option, DDL statements are extracted for all objects in the database. If this option is specified with the -m option, UPDATE statistics statements are extracted for all user created tables and indexes in the database. If neither -u nor -a is specified, the environment variable USER is used. On UNIX operating systems, this variable does not have to be explicitly set; on Windows systems, however, there is no default value for the USER environment variable: a user variable in the SYSTEM variables must be set, or a set USER=*username* must be issued for the session.

-m Generates the required UPDATE statements to replicate the statistics on tables, statistical views, columns and indexes.

-c When this option is specified in conjunction with the -m option, the db2look command does not generate COMMIT, CONNECT and CONNECT RESET statements. The default action is to generate these statements.

-r When this option is specified in conjunction with the -m option, the db2look command does not generate the RUNSTATS command. The default action is to generate the RUNSTATS command.

-l If this option is specified, then the db2look command will generate DDL for user defined table spaces, database partition groups and buffer pools. DDL for the following database objects is extracted when using the -l option:

- User-defined table spaces
- User-defined database partition groups
- User-defined buffer pools

-x If this option is specified, the db2look command will generate authorization DDL (GRANT statement, for example).

The supported authorizations include:

- Table: ALTER, SELECT, INSERT, DELETE, UPDATE, INDEX, REFERENCE, CONTROL
- View: SELECT, INSERT, DELETE, UPDATE, CONTROL
- Index: CONTROL
- Schema: CREATEIN, DROPIN, ALTERIN
- Database: CREATEDB, BINDADD, CONNECT, CREWATE_NOT_FENCED, IMPLICIT_SCHEMA
- User-defined function (UDF): EXECUTE
- User-defined method: EXECUTE
- Stored procedure: EXECUTE
- Package: CONTROL, BIND, EXECUTE
- Column: UPDATE, REFERENCES
- Table space: USE
- Sequence: USAGE, ALTER
- Workloads: USAGE
- Global variables
- Role
- Security labels
- Exemptions

-xd If this option is specified, the db2look command will generate all authorization DDLs, including authorization DDL for objects whose authorizations were granted by `SYSIBM` at object creation time.

-f Use this option to extract the configuration parameters and registry variables that affect the query optimizer.

-td *delimiter*
Specifies the statement delimiter for SQL statements generated by the db2look command. If this option is not specified, the default is the

semicolon (;). It is recommended that this option be used if the -e option is specified. In this case, the extracted objects might contain triggers or SQL routines.

-noview
If this option is specified, CREATE VIEW DDL statements will not be extracted.

-i *userid*
Use this option when working with a remote database.

-w *password*
Used with the -i option, this parameter allows the user to run the db2look command against a database that resides on a remote system. The user ID and the password are used by the db2look command to log on to the remote system. If working with remote databases, the remote database must be the same version as the local database. The db2look command does not have down-level or up-level support.

-wlm This option generates WLM specific DDL output, which can serve to generate CREATE and ALTER statements for:

- Histograms
- WLM Event Monitors
- Service Classes
- Workloads
- Thresholds
- Work Class Sets
- Work Action Sets

-wrapper *Wname*
Generates DDL statements for federated objects that apply to this wrapper. The federated DDL statements that might be generated include: CREATE WRAPPER, CREATE SERVER, CREATE USER MAPPING, CREATE NICKNAME, CREATE TYPE MAPPING, CREATE FUNCTION ... AS TEMPLATE, CREATE FUNCTION MAPPING, CREATE INDEX SPECIFICATION, and GRANT (privileges to nicknames, servers, indexes). Only one wrapper name is supported; an error is returned if less than one or more than one is specified. This option does not support non-relational data sources.

-server *Sname*
Generates DDL statements for federated objects that apply to this server. The federated DDL statements that might be generated include: CREATE WRAPPER, CREATE SERVER, CREATE USER MAPPING, CREATE NICKNAME, CREATE TYPE MAPPING, CREATE FUNCTION ... AS TEMPLATE, CREATE FUNCTION MAPPING, CREATE INDEX SPECIFICATION, and GRANT (privileges to nicknames, servers, indexes). Only one server name is supported; an error is returned if less than one or more than one is specified. This option does not support non-relational data sources.

-nofed Specifies that no federated DDL statements will be generated. When this option is specified, the -wrapper and -server options are ignored.

-fedonly
Specifies that only federated DDL statements will be generated.

-xs Exports all files necessary to register XML schemas and DTDs at the target

database, and generates appropriate commands for registering them. The set of XSR objects that will be exported is controlled by the -u, -z, and -a options.

-xdir *dirname*
Places exported XML-related files into the given path. If this option is not specified, all XML-related files will be exported into the current directory.

Examples

- Generate the DDL statements for objects created by user `walid` in database DEPARTMENT. The db2look output is sent to file `db2look.sql`:

```
db2look -d department -u walid -e -o db2look.sql
```

- Generate the DDL statements for objects that have schema name `ianhe`, created by user `walid`, in database DEPARTMENT. The db2look output is sent to file `db2look.sql`:

```
db2look -d department -u walid -z ianhe -e -o db2look.sql
```

- Generate the UPDATE statements to replicate the statistics for the database objects created by user `walid` in database DEPARTMENT. The output is sent to file `db2look.sql`:

```
db2look -d department -u walid -m -o db2look.sql
```

- Generate both the DDL statements for the objects created by user `walid` and the UPDATE statements to replicate the statistics on the database objects created by the same user. The db2look output is sent to file `db2look.sql`:

```
db2look -d department -u walid -e -m -o db2look.sql
```

- Generate the DDL statements for objects created by all users in the database DEPARTMENT. The db2look output is sent to file `db2look.sql`:

```
db2look -d department -a -e -o db2look.sql
```

- Generate the DDL statements for all user-defined database partition groups, buffer pools and table spaces. The db2look output is sent to file `db2look.sql`:

```
db2look -d department -l -o db2look.sql
```

- Generate the UPDATE statements for optimizer-related database and database manager configuration parameters, as well as the db2set statements for optimizer-related registry variables in database DEPARTMENT. The db2look output is sent to file `db2look.sql`:

```
db2look -d department -f -o db2look.sql
```

- Generate the DDL for all objects in database DEPARTMENT, the UPDATE statements to replicate the statistics on all tables and indexes in database DEPARTMENT, the GRANT authorization statements, the UPDATE statements for optimizer-related database and database manager configuration parameters, the db2set statements for optimizer-related registry variables, and the DDL for all user-defined database partition groups, buffer pools and table spaces in database DEPARTMENT. The output is sent to file `db2look.sql`.

```
db2look -d department -a -e -m -l -x -f -o db2look.sql
```

- Generate all authorization DDL statements for all objects in database DEPARTMENT, including the objects created by the original creator. (In this case, the authorizations were granted by SYSIBM at object creation time.) The db2look output is sent to file `db2look.sql`:

```
db2look -d department -xd -o db2look.sql
```

- Generate the DDL statements for objects created by all users in the database DEPARTMENT. The db2look output is sent to file `db2look.sql`:

```
db2look -d department -a -e -td % -o db2look.sql
```

The output can then be read by the CLP:

```
db2 -td% -f db2look.sql
```

- Generate the DDL statements for objects in database DEPARTMENT, excluding the CREATE VIEW statements. The db2look output is sent to file db2look.sql:

  ```
  db2look -d department -e -noview -o db2look.sql
  ```

- Generate the DDL statements for objects in database DEPARTMENT related to specified tables. The db2look output is sent to file db2look.sql:

  ```
  db2look -d department -e -t tab1 \"My TaBlE2\" -o db2look.sql
  ```

- Generate the DDL statements for all objects (federated and non-federated) in the federated database FEDDEPART. For federated DDL statements, only those that apply to the specified wrapper, FEDWRAP, are generated. The db2look output is sent to standard output:

  ```
  db2look -d feddepart -e -wrapper fedwrap
  ```

- Generate a script file that includes only non-federated DDL statements. The following system command can be run against a federated database (FEDDEPART) and yet only produce output like that found when run against a database which is not federated. The db2look output is sent to a file out.sql:

  ```
  db2look -d feddepart -e -nofed -o out
  ```

- Generate the DDL statements for objects that have schema name walid in the database DEPARTMENT. The files required to register any included XML schemas and DTDs are exported to the current directory. The db2look output is sent to file db2look.sql:

  ```
  db2look -d department -z walid -e -xs -o db2look.sql
  ```

- Generate the DDL statements for objects created by all users in the database DEPARTMENT. The files required to register any included XML schemas and DTDs are exported to directory /home/ofer/ofer/. The db2look output is sent to standard output:

  ```
  db2look -d department -a -e -xs -xdir /home/ofer/ofer/
  ```

- Generate WLM specific DDLs exclusively, in database DEPARTMENT.

  ```
  db2look -d department -wlm
  ```

  Generate the DDLs for all objects in the database DEPARTMENT.

  ```
  db2look -d department -wlm -e -l
  ```

Usage notes

On Windows operating systems, the db2look command must be run from a DB2 command window.

Several of the existing options support a federated environment. The following db2look command line options are used in a federated environment:

- -ap

  When used, AUDIT USING statements are generated.

- -e

  When used, federated DDL statements are generated.

- -x

  When used, GRANT statements are generated to grant privileges to the federated objects.

- -xd

  When used, federated DDL statements are generated to add system-granted privileges to the federated objects.

- -f

  When used, federated-related information is extracted from the database manager configuration.
- -m

  When used, statistics for nicknames are extracted.
- -wlm

  When used, WLM specific DDLs will be output.

The ability to use federated systems needs to be enabled in the database manager configuration in order to create federated DDL statements. After the db2look command generates the script file, you must set the **federated** configuration parameter to YES before running the script.

You need to modify the output script to add the remote passwords for the CREATE USER MAPPING statements.

You need to modify the db2look command output script by adding AUTHORIZATION and PASSWORD to those CREATE SERVER statements that are used to define a DB2 family instance as a data source.

Usage of the -tw option is as follows:

- To both generate the DDL statements for objects in the DEPARTMENT database associated with tables that have names beginning with `abc` and send the output to the db2look.sql file:

  ```
  db2look -d department -e -tw abc% -o db2look.sql
  ```
- To generate the DDL statements for objects in the DEPARTMENT database associated with tables that have a `d` as the second character of the name and to send the output to the db2look.sql file:

  ```
  db2look -d department -e -tw _d% -o db2look.sql
  ```
- The db2look command uses the LIKE predicate when evaluating which table names match the pattern specified by the *Tname* argument. Because the LIKE predicate is used, if either the _ character or the % character is part of the table name, the backslash (\) escape character must be used immediately before the _ or the %. In this situation, neither the _ nor the % can be used as a wildcard character in *Tname*. For example, to generate the DDL statements for objects in the DEPARTMENT database associated with tables that have a percent sign in the neither the first nor the last position of the name:

  ```
  db2look -d department -e -tw string\%string
  ```
- Case-sensitive, DBCS, and multi-word table and view names must be enclosed by both a backslash and double quotation marks. For example:

  ```
  \"My TabLe\"
  ```

  If a multibyte character set (MBCS) or double-byte character set (DBCS) name is not enclosed by the backward slash and double quotation delimiter and if it contains the same byte as the lowercase character, it will be converted into uppercase and db2look will look for a database object with the converted name. As a result, the DDL statement will not be extracted.
- The -tw option can be used with the -x option (to generate GRANT privileges), the -m option (to return table and column statistics), and the -l option (to generate the DDL for user-defined table spaces, database partition groups, and buffer pools). If the -t option is specified with the -tw option, the -t option (and its associated *Tname* argument) is ignored.

- The -tw option cannot be used to generate the DDL for tables (and their associated objects) that reside on federated data sources, or on DB2 Universal Database for z/OS and OS/390, DB2 for i5/OS® , or DB2 Server for VSE & VM.
- The -tw option is only supported via the CLP.

When requesting DDL on systems using the database partitioning feature, a warning message will be displayed in place of the DDL for table spaces that exist on inactive database partitions. To ensure proper DDL is produced for all table spaces all database partitions must be activated.

When extracting DDL for security label components of type array, the extracted DDL may not generate a security label component whose internal representation (i.e., the encoding of elements in that array) exactly matches the internal representation of that security label component within the database that the db2look extract was taken from. This can happen when a security label component of type array has been altered and one or more elements were added to it. In such cases, data extracted from one table and moved to another table, created from db2look output, will not have corresponding security label values, such that the protection of the new table may be compromised.

Related information

Nickname column and index names

Changing applications for migration

Chapter 6. File formats and data types

Export/Import/Load utility file formats

Five operating system file formats supported by the DB2 export, import, and load utilities are described:

DEL Delimited ASCII, for data exchange among a wide variety of database managers and file managers. This common approach to storing data uses special character delimiters to separate column values.

ASC Non-delimited ASCII, for importing or loading data from other applications that create flat text files with aligned column data.

PC/IXF
PC version of the Integration Exchange Format (IXF), the preferred method for data exchange within the database manager. PC/IXF is a structured description of a database table that contains an external representation of the internal table.

WSF Worksheet format, for data exchange with products such as Lotus 1-2-3 and Symphony. The load utility does not support this file format.

CURSOR
A cursor declared against an SQL query. This file type is only supported by the load utility.

When using DEL, WSF, or ASC data file formats, define the table, including its column names and data types, before importing the file. The data types in the operating system file fields are converted into the corresponding type of data in the database table. The import utility accepts data with minor incompatibility problems, including character data imported with possible padding or truncation, and numeric data imported into different types of numeric fields.

When using the PC/IXF data file format, the table does not need to exist before you begin the import operation. However, the user-defined distinct type (UDT) does need to be defined, otherwise you receive an undefined name error (SQL0204N). Similarly, when you are exporting to the PC/IXF data file format, UDTs are stored in the output file.

When using the CURSOR file type, the table, including its column names and data types, must be defined before beginning the load operation. The column types of the SQL query must be compatible with the corresponding column types in the target table. It is not necessary for the specified cursor to be open before starting the load operation. The load utility will process the entire result of the query associated with the specified cursor whether or not the cursor has been used to fetch rows.

Moving data across platforms - file format considerations

Compatibility is important when exporting, importing, or loading data across platforms. The following sections describe PC/IXF, delimited ASCII (DEL), and WSF file format considerations when moving data between different operating systems.

© Copyright IBM Corp. 1993, 2008

PC/IXF file format

PC/IXF is the recommended file format for transferring data across platforms. PC/IXF files allow the load utility or the import utility to process (normally machine dependent) numeric data in a machine-independent fashion. For example, numeric data is stored and handled differently by Intel® and other hardware architectures.

To provide compatibility of PC/IXF files among all products in the DB2 family, the export utility creates files with numeric data in Intel format, and the import utility expects it in this format.

Depending on the hardware platform, DB2 products convert numeric values between Intel and non-Intel formats (using byte reversal) during both export and import operations.

UNIX based implementations of DB2 database do not create multiple-part PC/IXF files during export. However, they will allow you to import a multiple-part PC/IXF file that was created by DB2. When importing this type of file, all parts should be in the same directory, otherwise an error is returned.

Single-part PC/IXF files created by UNIX based implementations of the DB2 export utility can be imported by DB2 database for Windows.

Delimited ASCII (DEL) file format

DEL files have differences based on the operating system on which they were created. The differences are:

- Row separator characters
  - UNIX based text files use a line feed (LF) character.
  - Non-UNIX based text files use a carriage return/line feed (CRLF) sequence.
- End-of-file character
  - UNIX based text files do not have an end-of-file character.
  - Non-UNIX based text files have an end-of-file character (X'1A').

Since DEL export files are text files, they can be transferred from one operating system to another. File transfer programs can handle operating system-dependant differences if you transfer the files in text mode; the conversion of row separator and end-of-file characters is not performed in binary mode.

Note: If character data fields contain row separator characters, these will also be converted during file transfer. This conversion causes unexpected changes to the data and, for this reason, it is recommended that you do not use DEL export files to move data across platforms. Use the PC/IXF file format instead.

WSF file format

Numeric data in WSF format files is stored using Intel machine format. This format allows Lotus WSF files to be transferred and used in different Lotus operating environments (for example, in Intel based and UNIX based systems).

As a result of this consistency in internal formats, exported WSF files from DB2 products can be used by Lotus 1-2-3 or Symphony running on a different platform. DB2 products can also import WSF files that were created on different platforms.

Transfer WSF files between operating systems in binary (not text) mode.

Note: Do not use the WSF file format to transfer data between DB2 databases on different platforms, because a loss of data can occur. Use the PC/IXF file format instead.

Delimited ASCII (DEL) file format

A Delimited ASCII (DEL) file is a sequential ASCII file with row and column delimiters. Each DEL file is a stream of ASCII characters consisting of cell values ordered by row, and then by column. Rows in the data stream are separated by row delimiters; within each row, individual cell values are separated by column delimiters.

The following table describes the format of DEL files that can be imported, or that can be generated as the result of an export action.

```
DEL file ::= Row 1 data || Row delimiter ||
             Row 2 data || Row delimiter ||
             .
             .
             .
             Row n data || Optional row delimiter

Row i data ::= Cell value(i,1) || Column delimiter ||
               Cell value(i,2) || Column delimiter ||
               .
               .
               .
               Cell value(i,m)

Row delimiter ::= ASCII line feed sequence[a]

Column delimiter ::= Default value ASCII comma (,)[b]

Cell value(i,j) ::= Leading spaces
                  || ASCII representation of a numeric value
                     (integer, decimal, or float)
                  || Delimited character string
                  || Non-delimited character string
                  || Trailing spaces

Non-delimited character string ::= A set of any characters except a
                                   row delimiter or a column delimiter

Delimited character string ::= A character string delimiter ||
                               An extended character string ||
                               A character string delimiter ||
                               Trailing garbage

Trailing garbage ::= A set of any characters except a row delimiter
                     or a column delimiter

Character string delimiter ::= Default value ASCII double quotation
                               marks (")[c]

extended character string ::= || A set of any characters except a
                                 row delimiter or a character string
                                 delimiter if the NODOUBLEDEL
                                 modifier is specified
                              || A set of any characters except a
                                 row delimiter or a character string
                                 delimiter if the character string
                                 is not part of two consecutive
```

```
                                  character string delimiters
                                || A set of any characters except a
                                  character string delimiter if the
                                  character string delimiter is not
                                  part of two consecutive character
                                  string delimiters, and the DELPRIORITYCHAR
                                  modifier is specified

End-of-file character ::= Hex '1A' (Windows operating system only)

ASCII representation of a numeric value[d] ::= Optional sign '+' or '-'
    || 1 to 31 decimal digits with an optional decimal point before,
       after, or between two digits
    || Optional exponent

Exponent ::= Character 'E' or 'e'
    || Optional sign '+' or '-'
    || 1 to 3 decimal digits with no decimal point

Decimal digit ::= Any one of the characters '0', '1', ... '9'

Decimal point ::= Default value ASCII period (.)[e]
```

- [a] The record delimiter is assumed to be a new line character, ASCII x0A. Data generated on the Windows operating system can use the carriage return/line feed 2-byte standard of 0x0D0A. Data in EBCDIC code pages should use the EBCDIC LF character (0x25) as the record delimiter (EBCDIC data can be loaded using the `codepage` file type modifier with the LOAD command).
- [b] The column delimiter can be specified with the `coldel` file type modifier.
- [c] The character string delimiter can be specified with the `chardel` file type modifier.

  Note: The default priority of delimiters is:

  1. Record delimiter
  2. Character delimiter
  3. Column delimiter
- [d] If the ASCII representation of a numeric value contains an exponent, it is a FLOAT constant. If it has a decimal point but no exponent, it is a DECIMAL constant. If it has no decimal point and no exponent, it is an INTEGER constant.
- [e] The decimal point character can be specified with the `decpt` file type modifier.

The export utility will replace every character string delimiter byte (default is double quote or x22) that is embedded within column data with two character string delimiter bytes (ie. doubling it). This is done so that the import parsing routines can distinguish between a character string delimiter byte that defines the beginning or end of a column, versus a character string delimiter byte embedded within the column data. Take caution when using an exported DEL file for some application other than the export utility, and note that the same doubling of character string delimiters occurs within 'FOR BIT' binary column data.

DEL data type descriptions

Table 50. Acceptable Data Type Forms for the DEL File Format

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
BIGINT	An INTEGER constant in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807.	ASCII representation of a numeric value in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807. Decimal and float numbers are truncated to integer values.
BLOB, CLOB	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is used as the database column value.
BLOB_FILE, CLOB_FILE	The character data for each BLOB/CLOB column is stored in individual files, and the file name is enclosed by character delimiters.	The delimited or non-delimited name of the file that holds the data.
CHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is truncated or padded with spaces (X'20'), if necessary, to match the width of the database column.
DATE	*yyyymmdd* (year month day) with no character delimiters. For example: 19931029 Alternatively, the DATESISO option can be used to specify that all date values are to be exported in ISO format.	A delimited or non-delimited character string containing a date value in an ISO format consistent with the territory code of the target database, or a non-delimited character string of the form *yyyymmdd*.
DBCLOB (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is used as the database column value.
DBCLOB_FILE (DBCS only)	The character data for each DBCLOB column is stored in individual files, and the file name is enclosed by character delimiters.	The delimited or non-delimited name of the file that holds the data.
DB2SECURITYLABEL	Column data is exported as "raw" data enclosed in quotation marks ("). Use the SECLABEL_TO_CHAR scalar function in the SELECT statement to convert the value to the security label string format.	The value in the data file is assumed by default to be the actual bytes that make up the internal representation of that security label. The value is assumed to be delimited by quotation marks (" ").

Table 50. Acceptable Data Type Forms for the DEL File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
DECIMAL	A DECIMAL constant with the precision and scale of the field being exported. The `decplusblank` file type modifier can be used to specify that positive decimal values are to be prefixed with a blank space instead of a plus sign (+).	ASCII representation of a numeric value that does not overflow the range of the database column into which the field is being imported. If the input value has more digits after the decimal point than can be accommodated by the database column, the excess digits are truncated.
FLOAT(long)	A FLOAT constant in the range -10E307 to 10E307.	ASCII representation of a numeric value in the range -10E307 to 10E307.
GRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is truncated or padded with double-byte spaces (for example, X'8140'), if necessary, to match the width of the database column.
INTEGER	An INTEGER constant in the range -2 147 483 648 to 2 147 483 647.	ASCII representation of a numeric value in the range -2 147 483 648 to 2 147 483 647. Decimal and float numbers are truncated to integer values.
LONG VARCHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is used as the database column value.
LONG VARGRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is used as the database column value.
SMALLINT	An INTEGER constant in the range -32 768 to 32 767.	ASCII representation of a numeric value in the range -32 768 to 32 767. Decimal and float numbers are truncated to integer values.
TIME	*hh.mm.ss* (hour minutes seconds). A time value in ISO format enclosed by character delimiters. For example: "09.39.43"	A delimited or non-delimited character string containing a time value in a format consistent with the territory code of the target database.

Table 50. Acceptable Data Type Forms for the DEL File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
TIMESTAMP	*yyyy-mm-dd-hh.mm.ss.nnnnnn* (year month day hour minutes seconds microseconds). A character string representing a date and time enclosed by character delimiters.	A delimited or non-delimited character string containing a time stamp value acceptable for storage in a database.
VARCHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is truncated, if necessary, to match the maximum width of the database column.
VARGRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is truncated, if necessary, to match the maximum width of the database column.

Example DEL file

Following is an example of a DEL file. Each line ends with a line feed sequence (on the Windows operating system, each line ends with a carriage return/line feed sequence).

```
"Smith, Bob",4973,15.46
"Jones, Bill",12345,16.34
"Williams, Sam",452,193.78
```

The following example illustrates the use of non-delimited character strings. The column delimiter has been changed to a semicolon, because the character data contains a comma.

```
Smith, Bob;4973;15.46
Jones, Bill;12345;16.34
Williams, Sam;452;193.78
```

Note:

1. A space (X'20') is never a valid delimiter.
2. Spaces that precede the first character, or that follow the last character of a cell value, are discarded during import. Spaces that are embedded in a cell value are not discarded.
3. A period (.) is not a valid character string delimiter, because it conflicts with periods in time stamp values.
4. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive.
5. For DEL data specified in an EBCDIC code page, the delimiters might not coincide with the shift-in and shift-out DBCS characters.
6. On the Windows operating system, the first occurrence of an end-of-file character (X'1A') that is not within character delimiters indicates the end-of-file. Any subsequent data is not imported.

7. A null value is indicated by the absence of a cell value where one would normally occur, or by a string of spaces.
8. Since some products restrict character fields to 254 or 255 bytes, the export utility generates a warning message whenever a character column of maximum length greater than 254 bytes is selected for export. The import utility accommodates fields that are as long as the longest LONG VARCHAR and LONG VARGRAPHIC columns.

Delimiter considerations for moving data

When moving delimited ASCII (DEL) files, it is important to ensure that the data being moved is not unintentionally altered because of problems with delimiter character recognition. To help prevent these errors, DB2 enforces several restrictions and provides a number of file type modifiers.

Delimiter restrictions

There are a number of restrictions in place that help prevent the chosen delimiter character from being treated as a part of the data being moved. First, delimiters are mutually exclusive. Second, a delimiter cannot be binary zero, a line-feed character, a carriage-return, or a blank space. As well, the default decimal point (.) cannot be a string delimiter. Finally, in a DBCS environment, the pipe (|) character delimiter is not supported.

The following characters are specified differently by an ASCII-family code page and an EBCDIC-family code page:

- The Shift-In (0x0F) and the Shift-Out (0x0E) character cannot be delimiters for an EBCDIC MBCS data file.
- Delimiters for MBCS, EUC, or DBCS code pages cannot be greater than 0x40, except the default decimal point for EBCDIC MBCS data, which is 0x4b.
- Default delimiters for data files in ASCII code pages or EBCDIC MBCS code pages are:
  - string delimiter: "(0x22, double quotation mark)
  - column delimiter: **,**(0x2c, comma)
- Default delimiters for data files in EBCDIC SBCS code pages are:
  - string delimiter: "(0x7F, double quotation mark)
  - column delimiter: **,**(0x6B, comma)
- The default decimal point for ASCII data files is 0x2e (period).
- The default decimal point for EBCDIC data files is 0x4B (period).
- If the code page of the server is different from the code page of the client, it is recommended that the hex representation of non-default delimiters be specified. For example,

```
db2 load from ... modified by chardel0x0C coldelX1e ...
```

Issues with delimiters during data movement

Double character delimiters

By default, for character-based fields of a DEL file, any instance of the character delimiter found within the field is represented by double character delimiters. For example, assuming that the character delimiter is the double quote, if you export the text `I am 6" tall.`, the output text in the DEL file reads `"I am 6"" tall."` Conversely, if the input text in a DEL file reads `"What a ""nice"" day!"`, the text is imported as `What a "nice" day!`

nodoubledel
Double character delimiter behavior can be disabled for the import, export, and load utilities by specifying the `nodoubledel` file type modifier. However, be aware that double character delimiter behavior exists in order to avoid parsing errors. When you use `nodoubledel` with export, the character delimiter is not doubled if it is present in character fields. When you use `nodoubledel` with import and load, the double character delimiter is not interpreted as a literal instance of the character delimiter.

nochardel
When you use the `nochardel` file type modifier with export, the character fields are not surrounded by character delimiters. When `nochardel` is used import and load, the character delimiters are not treated as special characters and are interpreted as actual data.

chardel
Other file type modifiers can be used to manually prevent confusion between default delimiters and the data. The`chardel` file type modifier specifies `x`, a single character, as the character string delimiter to be used instead of double quotation marks (as is the default).

coldel
Similarly, if you wanted to avoid using the default comma as a column delimiter, you could use `coldel`, which specifies `x`, a single character, as the column data delimiter.

delprioritychar
Another concern in regards to moving DEL files is maintaining the correct precedence order for delimiters. The default priority for delimiters is: row, character, column. However, some applications depend on the priority: character, row, column. For example, using the default priority, the DEL data file:

```
"Vincent <row delimiter> is a manager",<row delimiter>
```

would be interpreted as having two rows: `Vincent`, and `is a manager`, since <row delimiter> takes precedence over the character delimiter (″). Using `delprioritychar` gives the character delimiter (″) precedence over the row delimiter (`<row delimiter>`), meaning that the same DEL file would be interpreted (correctly) as having one row: `Vincent is a manager`.

Non-delimited ASCII (ASC) file format

The non-delimited ASCII format, known as ASC to the import and load utilities, comes in two varieties: fixed length and flexible length. For fixed length ASC, all records are of a fixed length. For flexible length ASC, records are delimited by a row delimiter (always a new line). The term *non-delimited* in non-delimited ASCII means that column values are not separated by delimiters.

When importing or loading ASC data, specifying the `reclen` file type modifier will indicate that the datafile is fixed length ASC. Not specifying it means that the datafile is flexible length ASC.

The non-delimited ASCII format, can be used for data exchange with any ASCII product that has a columnar format for data, including word processors. Each ASC file is a stream of ASCII characters consisting of data values ordered by row and column. Rows in the data stream are separated by row delimiters. Each column

within a row is defined by a beginning-ending location pair (specified by IMPORT parameters). Each pair represents locations within a row specified as byte positions. The first position within a row is byte position 1. The first element of each location pair is the byte on which the column begins, and the second element of each location pair is the byte on which the column ends. The columns might overlap. Every row in an ASC file has the same column definition.

An ASC file is defined by:

```
ASC file ::= Row 1 data || Row delimiter ||
             Row 2 data || Row delimiter ||
             .
             .
             .
             Row n data

Row i data ::= ASCII characters || Row delimiter

Row Delimiter ::= ASCII line feed sequence[a]
```

- [a] The record delimiter is assumed to be a new line character, ASCII x0A. Data generated on the Windows operating system can use the carriage return/line feed 2-byte standard of 0x0D0A. Data in EBCDIC code pages should use the EBCDIC LF character (0x25) as the record delimiter (EBCDIC data can be loaded using the `codepage` file type modifier with the LOAD command). The record delimiter is never interpreted to be part of a field of data.

ASC data type descriptions

Table 51. Acceptable Data Type Forms for the ASC File Format

Data Type	Form Acceptable to the Import Utility
BIGINT	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
BLOB/CLOB	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.
BLOB_FILE, CLOB_FILE, DBCLOB_FILE (DBCS only)	A delimited or non-delimited name of the file that holds the data.
CHAR	A string of characters. The character string is truncated or padded with spaces on the right, if necessary, to match the width of the target column.
DATE	A character string representing a date value in a format consistent with the territory code of the target database. The beginning and ending locations should specify a field width that is within the range for the external representation of a date.

Table 51. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
DBCLOB (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.
DECIMAL	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range of the database column into which they are being imported. If the input value has more digits after the decimal point than the scale of the database column, the excess digits are truncated. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
FLOAT(long)	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. All values are valid. A comma, period, or colon is considered to be a decimal point. An uppercase or lowercase E is accepted as the beginning of the exponent of a FLOAT constant. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
GRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated or padded with double-byte spaces (0x8140) on the right, if necessary, to match the maximum length of the target column.
INTEGER	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -2 147 483 648 to 2 147 483 647. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
LONG VARCHAR	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.
LONG VARGRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.

Table 51. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
SMALLINT	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -32 768 to 32 767. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
TIME	A character string representing a time value in a format consistent with the territory code of the target database. The beginning and ending locations should specify a field width that is within the range for the external representation of a time.
TIMESTAMP	A character string representing a time stamp value acceptable for storage in a database. The beginning and ending locations should specify a field width that is within the range for the external representation of a time stamp.
VARCHAR	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.
VARGRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.

Example ASC file

Following is an example of an ASC file. Each line ends with a line feed sequence (on the Windows operating system, each line ends with a carriage return/line feed sequence).

```
Smith, Bob       4973     15.46
Jones, Suzanne   12345    16.34
Williams, Sam    452123  193.78
```

Note:

1. ASC files are assumed not to contain column names.
2. Character strings are *not* enclosed by delimiters. The data type of a column in the ASC file is determined by the data type of the target column in the database table.
3. A NULL is imported into a nullable database column if:
   - A field of blanks is targeted for a numeric, DATE, TIME, or TIMESTAMP database column
   - A field with no beginning and ending location pairs is specified
   - A location pair with beginning and ending locations equal to zero is specified

- A row of data is too short to contain a valid value for the target column
- The NULL INDICATORS load option is used, and an `N` (or other value specified by the user) is found in the null indicator column.

4. If the target column is not nullable, an attempt to import a field of blanks into a numeric, DATE, TIME, or TIMESTAMP column causes the row to be rejected.
5. If the input data is not compatible with the target column, and that column is nullable, a null is imported or the row is rejected, depending on where the error is detected. If the column is not nullable, the row is rejected. Messages are written to the message file, specifying incompatibilities that are found.

PC Version of IXF file format

The PC version of IXF (PC/IXF) file format is a database manager adaptation of the Integration Exchange Format (IXF) data interchange architecture. The IXF architecture was specifically designed to enable the exchange of relational database structures and data. The PC/IXF architecture allows the database manager to export a database without having to anticipate the requirements and idiosyncrasies of a receiving product. Similarly, a product importing a PC/IXF file need only understand the PC/IXF architecture; the characteristics of the product which exported the file are not relevant. The PC/IXF file architecture maintains the independence of both the exporting and the importing database systems.

The IXF architecture is a generic relational database exchange format that supports a rich set of relational data types, including some types that might not be supported by specific relational database products. The PC/IXF file format preserves this flexibility; for example, the PC/IXF architecture supports both single-byte character string (SBCS) and double-byte character string (DBCS) data types. Not all implementations support all PC/IXF data types; however, even restricted implementations provide for the detection and disposition of unsupported data types during import.

In general, a PC/IXF file consists of an unbroken sequence of variable-length records. The file contains the following record types in the order shown:

- One header record of record type H
- One table record of record type T
- Multiple column descriptor records of record type C (one record for each column in the table)
- Multiple data records of record type D (each row in the table is represented by one or more D records).

A PC/IXF file might also contain application records of record type A, anywhere after the H record. These records are permitted in PC/IXF files to enable an application to include additional data, not defined by the PC/IXF format, in a PC/IXF file. A records are ignored by any program reading a PC/IXF file that does not have particular knowledge about the data format and content implied by the application identifier in the A record.

Every record in a PC/IXF file begins with a record length indicator. This is a 6-byte right justified character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Programs reading PC/IXF files should use these record lengths to locate the end of the current record and the beginning of the next record. H, T, and C records must be sufficiently large to include all of their defined fields, and, of course, their record length fields must agree with their

actual lengths. However, if extra data (for example, a *new* field), is added to the end of one of these records, pre-existing programs reading PC/IXF files should ignore the extra data, and generate no more than a warning message. Programs writing PC/IXF files, however, should write H, T and C records that are the precise length needed to contain all of the defined fields.

If a PC/IXF file contains LOB Location Specifier (LLS) columns, each LLS column must have its own D record. D records are automatically created by the export utility, but you will need to create them manually if you are using a third party tool to generate the PC/IXF files. Further, an LLS is required for each LOB column in a table, including those with a null value. If a LOB column is null, you will need to create an LLS representing a null LOB.

The D record entry for each XML column will contain two bytes little endian indicating the XML data specifier (XDS) length, followed by the XDS itself.

For example, the following XDS:

```
XDS FIL="a.xml" OFF="1000" LEN="100" SCH="RENATA.SCHEMA" />
```

will be represented by the following bytes in a D record:

```
0x3D 0x00 XDS FIL="a.xml" OFF="1000" LEN="100" SCH="RENATA.SCHEMA" />
```

PC/IXF file records are composed of fields which contain character data. The import and export utilities interpret this character data using the CPGID of the target database, with two exceptions:

- The IXFADATA field of A records.

  The code page environment of character data contained in an IXFADATA field is established by the application which creates and processes a particular A record; that is, the environment varies by implementation.
- The IXFDCOLS field of D records.

  The code page environment of character data contained in an IXFDCOLS field is a function of information contained in the C record which defines a particular column and its data.

Numeric fields in H, T, and C records, and in the prefix portion of D and A records should be right justified single-byte character representations of integer values, filled with leading zeros or blanks. A value of zero should be indicated with at least one (right justified) zero character, not blanks. Whenever one of these numeric fields is not used, for example IXFCLENG, where the length is implied by the data type, it should be filled with blanks. These numeric fields are:

```
IXFHRECL, IXFTRECL, IXFCRECL, IXFDRECL, IXFARECL,
IXFHHCNT, IXFHSBCP, IXFHDBCP, IXFTCCNT, IXFTNAML,
IXFCLENG, IXFCDRID, IXFCPOSN, IXFCNAML, IXFCTYPE,
IXFCSBCP, IXFCDBCP, IXFCNDIM, IXFCDSIZ, IXFDRID
```

Note: The database manager PC/IXF file format is not identical to the System/370™.

PC/IXF record types

There are five basic PC/IXF record types:

- header
- table
- column descriptor

- data
- application

and six application subtypes that DB2 uses:

- index
- hierarchy
- subtable
- continuation
- terminate
- identity

Each PC/IXF record type is defined as a sequence of fields; these fields are required, and must appear in the order shown.

HEADER RECORD

FIELD NAME	LENGTH	TYPE	COMMENTS
IXFHRECL	06-BYTE	CHARACTER	record length
IXFHRECT	01-BYTE	CHARACTER	record type = 'H'
IXFHID	03-BYTE	CHARACTER	IXF identifier
IXFHVERS	04-BYTE	CHARACTER	IXF version
IXFHPROD	12-BYTE	CHARACTER	product
IXFHDATE	08-BYTE	CHARACTER	date written
IXFHTIME	06-BYTE	CHARACTER	time written
IXFHHCNT	05-BYTE	CHARACTER	heading record count
IXFHSBCP	05-BYTE	CHARACTER	single byte code page
IXFHDBCP	05-BYTE	CHARACTER	double byte code page
IXFHFIL1	02-BYTE	CHARACTER	reserved

The following fields are contained in the header record:

IXFHRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The H record must be sufficiently long to include all of its defined fields.

IXFHRECT
The IXF record type, which is set to H for this record.

IXFHID
The file format identifier, which is set to IXF for this file.

IXFHVERS
The PC/IXF format level used when the file was created, which is set to '0002'.

IXFHPROD
A field that can be used by the program creating the file to identify itself. If this field is filled in, the first six bytes are used to identify the product creating the file, and the last six bytes are used to indicate the version or release of the creating product. The database manager uses this field to signal the existence of database manager-specific data.

IXFHDATE
The date on which the file was written, in the form *yyyymmdd*.

IXFHTIME
The time at which the file was written, in the form *hhmmss*. This field is optional and can be left blank.

IXFHHCNT
The number of H, T, and C records in this file that precede the first data record. A records are not included in this count.

IXFHSBCP
Single-byte code page field, containing a single-byte character representation of a SBCS CPGID or '00000'.

The export utility sets this field equal to the SBCS CPGID of the exported database table. For example, if the table SBCS CPGID is 850, this field contains '00850'.

IXFHDBCP
Double-byte code page field, containing a single-byte character representation of a DBCS CPGID or '00000'.

The export utility sets this field equal to the DBCS CPGID of the exported database table. For example, if the table DBCS CPGID is 301, this field contains '00301'.

IXFHFIL1
Spare field set to two blanks to match a reserved field in host IXF files.

TABLE RECORD

FIELD NAME	LENGTH	TYPE	COMMENTS
IXFTRECL	006-BYTE	CHARACTER	record length
IXFTRECT	001-BYTE	CHARACTER	record type = 'T'
IXFTNAML	003-BYTE	CHARACTER	name length
IXFTNAME	256-BYTE	CHARACTER	name of data
IXFTQULL	003-BYTE	CHARACTER	qualifier length
IXFTQUAL	256-BYTE	CHARACTER	qualifier
IXFTSRC	012-BYTE	CHARACTER	data source
IXFTDATA	001-BYTE	CHARACTER	data convention = 'C'
IXFTFORM	001-BYTE	CHARACTER	data format = 'M'
IXFTMFRM	005-BYTE	CHARACTER	machine format = 'PC'
IXFTLOC	001-BYTE	CHARACTER	data location = 'I'
IXFTCCNT	005-BYTE	CHARACTER	'C' record count
IXFTFIL1	002-BYTE	CHARACTER	reserved
IXFTDESC	030-BYTE	CHARACTER	data description
IXFTPKNM	257-BYTE	CHARACTER	primary key name
IXFTDSPC	257-BYTE	CHARACTER	reserved
IXFTISPC	257-BYTE	CHARACTER	reserved
IXFTLSPC	257-BYTE	CHARACTER	reserved

The following fields are contained in the table record:

IXFTRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The T record must be sufficiently long to include all of its defined fields.

IXFTRECT
The IXF record type, which is set to T for this record.

IXFTNAML
The length, in bytes, of the table name in the IXFTNAME field.

IXFTNAME
The name of the table. If each file has only one table, this is an informational field only. The database manager does not use this field when importing data. When writing a PC/IXF file, the database manager writes the DOS file name (and possibly path information) to this field.

IXFTQULL
The length, in bytes, of the table name qualifier in the IXFTQUAL field.

IXFTQUAL
Table name qualifier, which identifies the creator of a table in a relational system. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFTSRC
Used to indicate the original source of the data. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFTDATA
Convention used to describe the data. This field must be set to `C` for import and export, indicating that individual column attributes are described in the following column descriptor (C) records, and that data follows PC/IXF conventions.

IXFTFORM
Convention used to store numeric data. This field must be set to `M`, indicating that numeric data in the data (D) records is stored in the machine (internal) format specified by the IXFTMFRM field.

IXFTMFRM
The format of any machine data in the PC/IXF file. The database manager will only read or write files if this field is set to `PC`*bbb*, where *b* represents a blank, and PC specifies that data in the PC/IXF file is in IBM PC machine format.

IXFTLOC
The location of the data. The database manager only supports a value of I, meaning the data is internal to this file.

IXFTCCNT
The number of C records in this table. It is a right-justified character representation of an integer value.

IXFTFIL1
Spare field set to two blanks to match a reserved field in host IXF files.

IXFTDESC
Descriptive data about the table. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field. This field contains `NOT NULL WITH DEFAULT` if the column was not null with default, and the table name came from a workstation database.

IXFTPKNM
The name of the primary key defined on the table (if any). The name is stored as a null-terminated string.

IXFTDSPC
This field is reserved for future use.

IXFTISPC
This field is reserved for future use.

IXFTLSPC
This field is reserved for future use.

COLUMN DESCRIPTOR RECORD

FIELD NAME	LENGTH	TYPE	COMMENTS
IXFCRECL	006-BYTE	CHARACTER	record length
IXFCRECT	001-BYTE	CHARACTER	record type = 'C'
IXFCNAML	003-BYTE	CHARACTER	column name length
IXFCNAME	256-BYTE	CHARACTER	column name
IXFCNULL	001-BYTE	CHARACTER	column allows nulls
IXFCDEF	001-BYTE	CHARACTER	column has defaults
IXFCSLCT	001-BYTE	CHARACTER	column selected flag
IXFCKPOS	002-BYTE	CHARACTER	position in primary key
IXFCCLAS	001-BYTE	CHARACTER	data class
IXFCTYPE	003-BYTE	CHARACTER	data type
IXFCSBCP	005-BYTE	CHARACTER	single byte code page
IXFCDBCP	005-BYTE	CHARACTER	double byte code page
IXFCLENG	005-BYTE	CHARACTER	column data length
IXFCDRID	003-BYTE	CHARACTER	'D' record identifier
IXFCPOSN	006-BYTE	CHARACTER	column position
IXFCDESC	030-BYTE	CHARACTER	column description
IXFCLOBL	020-BYTE	CHARACTER	lob column length
IXFCUDTL	003-BYTE	CHARACTER	UDT name length
IXFCUDTN	256-BYTE	CHARACTER	UDT name
IXFCDEFL	003-BYTE	CHARACTER	default value length
IXFCDEFV	254-BYTE	CHARACTER	default value
IXFCREF	001-BYTE	CHARACTER	reference type
IXFCNDIM	002-BYTE	CHARACTER	number of dimensions
IXFCDSIZ	varying	CHARACTER	size of each dimension

The following fields are contained in column descriptor records:

IXFCRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The C record must be sufficiently long to include all of its defined fields.

IXFCRECT
The IXF record type, which is set to C for this record.

IXFCNAML
The length, in bytes, of the column name in the IXFCNAME field.

IXFCNAME
The name of the column.

IXFCNULL
Specifies if nulls are permitted in this column. Valid settings are `Y` or `N`.

IXFCDEF
Specifies if a default value is defined for this field. Valid settings are `Y` or `N`.

IXFCSLCT
An obsolete field whose intended purpose was to allow selection of a subset of columns in the data. Programs writing PC/IXF files should always store a Y in this field. Programs reading PC/IXF files should ignore the field.

IXFCKPOS
The position of the column as part of the primary key. Valid values range from `01` to `16`, or `N` if the column is not part of the primary key.

IXFCCLAS
The class of data types to be used in the IXFCTYPE field. The database manager only supports relational types (`R`).

IXFCTYPE
The data type for the column.

IXFCSBCP
Contains a single-byte character representation of a SBCS CPGID. This field specifies the CPGID for single-byte character data, which occurs with the IXFDCOLS field of the D records for this column.

The semantics of this field vary with the data type for the column (specified in the IXFCTYPE field).

- For a character string column, this field should normally contain a non-zero value equal to that of the IXFHSBCP field in the H record; however, other values are permitted. If this value is zero, the column is interpreted to contain bit string data.
- For a numeric column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a date or time column, this field is not meaningful. It is set to the value of the IXFHSBCP field by the export utility, and ignored by the import utility.
- For a graphic column, this field must be zero.

IXFCDBCP
Contains a single-byte character representation of a DBCS CPGID. This field specifies the CPGID for double-byte character data, which occurs with the IXFDCOLS field of the D records for this column.

The semantics of this field vary with the data type for the column (specified in the IXFCTYPE field).

- For a character string column, this field should either be zero, or contain a value equal to that of the IXFHDBCP field in the H record; however, other values are permitted. If the value in the IXFCSBCP field is zero, the value in this field must be zero.
- For a numeric column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a date or time column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a graphic column, this field must have a value equal to the value of the IXFHDBCP field.

IXFCLENG
Provides information about the size of the column being described. For some data types, this field is unused, and should contain blanks. For other data types, this field contains the right-justified character representation of an integer specifying the column length. For yet other data types, this field

is divided into two subfields: 3 bytes for precision, and 2 bytes for scale; both of these subfields are right-justified character representations of integers.

IXFCDRID
The D record identifier. This field contains the right-justified character representation of an integer value. Several D records can be used to contain each row of data in the PC/IXF file. This field specifies which D record (of the several D records contributing to a row of data) contains the data for the column. A value of one (for example, `001`) indicates that the data for a column is in the first D record in a row of data. The first C record must have an IXFCDRID value of one. All subsequent C records must have an IXFCDRID value equal to the value in the preceding C record, or one higher.

IXFCPOSN
The value in this field is used to locate the data for the column within one of the D records representing a row of table data. It is the starting position of the data for this column within the IXFDCOLS field of the D record. If the column is nullable, IXFCPOSN points to the null indicator; otherwise, it points to the data itself. If a column contains varying length data, the data itself begins with the current length indicator. The IXFCPOSN value for the first byte in the IXFDCOLS field of the D record is one (not zero). If a column is in a new D record, the value of IXFCPOSN should be one; otherwise, IXFCPOSN values should increase from column to column to such a degree that the data values do not overlap.

IXFCDESC
Descriptive information about the column. This is an informational field only. If a program writing to a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFCLOBL
The length, in bytes, of the long or the LOB defined in this column. If this column is not a long or a LOB, the value in this field is `000`.

IXFCUDTL
The length, in bytes, of the user defined type (UDT) name in the IXFCUDTN field. If the type of this column is not a UDT, the value in this field is `000`.

IXFCUDTN
The name of the user defined type that is used as the data type for this column.

IXFCDEFL
The length, in bytes, of the default value in the IXFCDEFV field. If this column does not have a default value, the value in this field is `000`.

IXFCDEFV
Specifies the default value for this column, if one has been defined.

IXFCREF
If the column is part of a hierarchy, this field specifies whether the column is a data column (`D`), or a reference column (`R`).

IXFCNDIM
The number of dimensions in the column. Arrays are not supported in this

version of PC/IXF. This field must therefore contain a character representation of a zero integer value.

IXFCDSIZ
The size or range of each dimension. The length of this field is five bytes per dimension. Since arrays are not supported (that is, the number of dimensions must be zero), this field has zero length, and does not actually exist.

DATA RECORD

FIELD NAME	LENGTH	TYPE	COMMENTS
IXFDRECL	06-BYTE	CHARACTER	record length
IXFDRECT	01-BYTE	CHARACTER	record type = 'D'
IXFDRID	03-BYTE	CHARACTER	'D' record identifier
IXFDFIL1	04-BYTE	CHARACTER	reserved
IXFDCOLS	varying	variable	columnar data

The following fields are contained in the data records:

IXFDRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each D record must be sufficiently long to include all significant data for the current occurrence of the last data column stored in the record.

IXFDRECT
The IXF record type, which is set to D for this record, indicating that it contains data values for the table.

IXFDRID
The record identifier, which identifies a particular D record within the sequence of several D records contributing to a row of data. For the first D record in a row of data, this field has a value of one; for the second D record in a row of data, this field has a value of two, and so on. In each row of data, all the D record identifiers called out in the C records must actually exist.

IXFDFIL1
Spare field set to four blanks to match reserved fields, and hold a place for a possible shift-out character, in host IXF files.

IXFDCOLS
The area for columnar data. The data area of a data record (D record) is composed of one or more column entries. There is one column entry for each column descriptor record, which has the same D record identifier as the D record. In the D record, the starting position of the column entries is indicated by the IXFCPOSN value in the C records.

The format of the column entry data depends on whether or not the column is nullable:

- If the column is nullable (the IXFCNULL field is set to Y), the column entry data includes a null indicator. If the column is not null, the indicator is followed by data type-specific information, including the actual database value. The null indicator is a two-byte value set to x'0000' for not null, and x'FFFF' for null.
- If the column is not nullable, the column entry data includes only data type-specific information, including the actual database value.

For varying-length data types, the data type-specific information includes a current length indicator. The current length indicators are 2-byte integers in a form specified by the IXFTMFRM field.

The length of the data area of a D record cannot exceed 32 771 bytes.

```
APPLICATION RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFARECL       06-BYTE   CHARACTER   record length
   IXFARECT       01-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       12-BYTE   CHARACTER   application identifier
   IXFADATA       varying   variable    application-specific data
```

The following fields are contained in application records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies the application creating the A record. PC/IXF files created by the database manager can have A records with the first 6 characters of this field set to a constant identifying the database manager, and the last 6 characters identifying the release or version of the database manager or another application writing the A record.

IXFADATA
This field contains application dependent supplemental data, whose form and content are known only to the program creating the A record, and to other applications which are likely to process the A record.

```
DB2 INDEX RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                            'DB2    02.00'
   IXFAITYP       001-BYTE   CHARACTER   application specific data type =
                                            'I'
   IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
   IXFANDXL       002-BYTE   SHORT INT   length of name of the index
   IXFANDXN       256-BYTE   CHARACTER   name of the index
   IXFANCL        002-BYTE   SHORT INT   length of name of the index creator
   IXFANCN        256-BYTE   CHARACTER   name of the index creator
   IXFATABL       002-BYTE   SHORT INT   length of name of the table
   IXFATABN       256-BYTE   CHARACTER   name of the table
   IXFATCL        002-BYTE   SHORT INT   length of name of the table creator
   IXFATCN        256-BYTE   CHARACTER   name of the table creator
   IXFAUNIQ       001-BYTE   CHARACTER   unique rule
   IXFACCNT       002-BYTE   CHARACTER   column count
```

```
IXFAREVS       001-BYTE   CHARACTER   allow reverse scan flag
IXFAPCTF       002-BYTE   CHARACTER   amount of pct free
IXFAPCTU       002-BYTE   CHARACTER   amount of minpctused
IXFAEXTI       001-BYTE   CHARACTER   reserved
IXFACNML       002-BYTE   SHORT INT   length of name of the columns
IXFACOLN       varying    CHARACTER   name of the columns in the index
```

One record of this type is specified for each user defined index. This record is located after all of the C records for the table. The following fields are contained in DB2 index records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFAITYP
Specifies that this is subtype ″I″ of DB2 application records.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFANDXL
The length, in bytes, of the index name in the IXFANDXN field.

IXFANDXN
The name of the index.

IXFANCL
The length, in bytes, of the index creator name in the IXFANCN field.

IXFANCN
The name of the index creator.

IXFATABL
The length, in bytes, of the table name in the IXFATABN field.

IXFATABN
The name of the table.

IXFATCL
The length, in bytes, of the table creator name in the IXFATCN field.

IXFATCN
The name of the table creator.

IXFAUNIQ
Specifies the type of index. Valid values are `P` for a primary key, `U` for a unique index, and `D` for a non unique index.

IXFACCNT
Specifies the number of columns in the index definition.

IXFAREVS
Specifies whether reverse scan is allowed on this index. Valid values are `Y` for reverse scan, and `N` for no reverse scan.

IXFAPCTF
Specifies the percentage of index pages to leave as free. Valid values range from `-1` to `99`. If a value of `-1` or zero is specified, the system default value is used.

IXFAPCTU
Specifies the minimum percentage of index pages that must be free before two index pages can be merged. Valid values range from `00` to `99`.

IXFAEXTI
Reserved for future use.

IXFACNML
The length, in bytes, of the column names in the IXFACOLN field.

IXFACOLN
The names of the columns that are part of this index. Valid values are in the form *+name-name...*, where + specifies an ascending sort on the column, and - specifies a descending sort on the column.

```
DB2 HIERARCHY RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                            'DB2    02.00'
   IXFAXTYP       001-BYTE   CHARACTER   application specific data type =
                                            'X'
   IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
   IXFAYCNT       010-BYTE   CHARACTER   'Y' record count for this hierarchy
   IXFAYSTR       010-BYTE   CHARACTER   starting column of this hierarchy
```

One record of this type is used to describe a hierarchy. All subtable records (see below) must be located immediately after the hierarchy record, and hierarchy records are located after all of the C records for the table. The following fields are contained in DB2 hierarchy records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFAXTYP
Specifies that this is subtype ″X″ of DB2 application records.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFAYCNT
Specifies the number of subtable records that are expected after this hierarchy record.

IXFAYSTR
Specifies the index of the subtable records at the beginning of the exported data. If export of a hierarchy was started from a non-root subtable, all parent tables of this subtable are exported. The position of this subtable inside of the IXF file is also stored in this field. The first X record represents the column with an index of zero.

DB2 SUBTABLE RECORD

FIELD NAME	LENGTH	TYPE	COMMENTS
IXFARECL	006-BYTE	CHARACTER	record length
IXFARECT	001-BYTE	CHARACTER	record type = 'A'
IXFAPPID	012-BYTE	CHARACTER	application identifier = 'DB2 02.00'
IXFAYTYP	001-BYTE	CHARACTER	application specific data type = 'Y'
IXFADATE	008-BYTE	CHARACTER	date written from the 'H' record
IXFATIME	006-BYTE	CHARACTER	time written from the 'H' record
IXFASCHL	003-BYTE	CHARACTER	type schema name length
IXFASCHN	256-BYTE	CHARACTER	type schema name
IXFATYPL	003-BYTE	CHARACTER	type name length
IXFATYPN	256-BYTE	CHARACTER	type name
IXFATABL	003-BYTE	CHARACTER	table name length
IXFATABN	256-BYTE	CHARACTER	table name
IXFAPNDX	010-BYTE	CHARACTER	subtable index of parent table
IXFASNDX	005-BYTE	CHARACTER	starting column index of current table
IXFAENDX	005-BYTE	CHARACTER	ending column index of current table

One record of this type is used to describe a subtable as part of a hierarchy. All subtable records belonging to a hierarchy must be stored together, and immediately after the corresponding hierarchy record. A subtable is composed of one or more columns, and each column is described in a column record. Each column in a subtable must be described in a consecutive set of C records. The following fields are contained in DB2 subtable records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFAYTYP
Specifies that this is subtype ″Y″ of DB2 application records.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFASCHL
The length, in bytes, of the subtable schema name in the IXFASCHN field.

IXFASCHN
The name of the subtable schema.

IXFATYPL
The length, in bytes, of the subtable name in the IXFATYPN field.

IXFATYPN
The name of the subtable.

IXFATABL
The length, in bytes, of the table name in the IXFATABN field.

IXFATABN
The name of the table.

IXFAPNDX
Subtable record index of the parent subtable. If this subtable is the root of a hierarchy, this field contains the value -1.

IXFASNDX
Starting index of the column records that made up this subtable.

IXFAENDX
Ending index of the column records that made up this subtable.

```
DB2 CONTINUATION RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                            'DB2    02.00'
   IXFACTYP       001-BYTE   CHARACTER   application specific data type = 'C'
   IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
   IXFALAST       002-BYTE   SHORT INT   last diskette volume number
   IXFATHIS       002-BYTE   SHORT INT   this diskette volume number
   IXFANEXT       002-BYTE   SHORT INT   next diskette volume number
```

This record is found at the end of each file that is part of a multi-volume IXF file, unless that file is the final volume; it can also be found at the beginning of each file that is part of a multi-volume IXF file, unless that file is the first volume. The purpose of this record is to keep track of file order. The following fields are contained in DB2 continuation records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFACTYP
Specifies that this is subtype ″C″ of DB2 application records.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFALAST
This field is a binary field, in little-endian format. The value should be one less than the value in IXFATHIS.

IXFATHIS
This field is a binary field, in little-endian format. The value in this field on consecutive volumes should also be consecutive. The first volume has a value of 1.

IXFANEXT
This field is a binary field, in little-endian format. The value should be one more than the value in IXFATHIS, unless the record is at the beginning of the file, in which case the value should be zero.

```
DB2 TERMINATE RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                            'DB2    02.00'
   IXFAETYP       001-BYTE   CHARACTER   application specific data type =
                                            'E'
   IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
```

This record is the end-of-file marker found at the end of an IXF file. The following fields are contained in DB2 terminate records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFAETYP
Specifies that this is subtype ″E″ of DB2 application records.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

```
DB2 IDENTITY RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFARECL       06-BYTE   CHARACTER   record length
   IXFARECT       01-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       12-BYTE   CHARACTER   application identifier
   IXFATYPE       01-BYTE   CHARACTER   application specific record type = 'S'
   IXFADATE       08-BYTE   CHARACTER   application record creation date
   IXFATIME       06-BYTE   CHARACTER   application record creation time
   IXFACOLN       06-BYTE   CHARACTER   column number of the identity column
   IXFAITYP       01-BYTE   CHARACTER   generated always ('Y' or 'N')
   IXFASTRT       33-BYTE   CHARACTER   identity START AT value
   IXFAINCR       33-BYTE   CHARACTER   identity INCREMENT BY value
   IXFACACH       10-BYTE   CHARACTER   identity CACHE value
   IXFAMINV       33-BYTE   CHARACTER   identity MINVALUE
   IXFAMAXV       33-BYTE   CHARACTER   identity MAXVALUE
   IXFACYCL       01-BYTE   CHARACTER   identity CYCLE ('Y' or 'N')
   IXFAORDR       01-BYTE   CHARACTER   identity ORDER ('Y' or 'N')
   IXFARMRL       03-BYTE   CHARACTER   identity Remark length
   IXFARMRK      254-BYTE   CHARACTER   identity Remark value
```

The following fields are contained in DB2 identity records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFATYPE
Application specific record type. This field should always have a value of ″S″.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFACOLN
Column number of the identity column in the table.

IXFAITYP
The type of the identity column. A value of ″Y″ indicates that the identity column is always GENERATED. All other values are interpreted to mean that the column is of type GENERATED BY DEFAULT.

IXFASTRT
The START AT® value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFAINCR
The INCREMENT BY value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFACACH
The CACHE value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation. A value of ″1″ corresponds to the NO CACHE option.

IXFAMINV
The MINVALUE for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFAMAXV
The MAXVALUE for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFACYCL
The CYCLE value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation. A value of ″Y″ corresponds to the CYCLE option, any other value corresponds to NO CYCLE.

IXFAORDR
The ORDER value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation. A value of ″Y″ corresponds to the ORDER option, any other value corresponds to NO ORDER.

IXFARMRL
The length, in bytes, of the remark in IXFARMRK field.

IXFARMRK
This is the user-entered remark associated with the identity column. This is an informational field only. The database manager does not use this field when importing data.

PC/IXF data types

Table 52. PC/IXF Data Types

Name	IXFCTYPE Value	Description
BIGINT	492	An 8-byte integer in the form specified by IXFTMFRM. It represents a whole number between -9 223 372 036 854 775 808 and 9 223 372 036 854 775 807. IXFCSBCP and IXFCDBCP are not significant , and should be zero. IXFCLENG is not used, and should contain blanks.
BLOB, CLOB	404, 408	A variable-length character string. The maximum length of the string is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 32 767 bytes. The string itself is preceded by a current length indicator, which is a 4-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. The following applies to BLOBs only: If IXFCSBCP is zero, the string is bit data, and should not be translated by any transformation program. The following applies to CLOBs only: If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP.
BLOB_LOCATION_ SPECIFIER and DBCLOB_ LOCATION_ SPECIFIER	960, 964, 968	A fixed-length field, which cannot exceed 255 bytes. The LOB Location Specifier (LLS)is located in the code page indicated by IXFCSBCP. If IXFCSBCP is zero, the LLS is bit data and should not be translated by any transformation program. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. Since the length of the LLS is stored in IXFCLENG, the actual length of the original LOB is lost. PC/IXF files with columns of this type should not be used to recreate the LOB field since the LOB will be created with the length of the LLS.

Table 52. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
BLOB_FILE, CLOB_FILE, DBCLOB_FILE	916, 920, 924	A fixed-length field containing an SQLFILE structure with the *name_length* and the *name* fields filled in. The length of the structure is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 255 bytes. The file name is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the file name can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the file name is bit data and should not be translated by any transformation program. Since the length of the structure is stored in IXFCLENG, the actual length of the original LOB is lost. IXF files with columns of type BLOB_FILE, CLOB_FILE, or DBCLOB_FILE should not be used to recreate the LOB field, since the LOB will be created with a length of *sql_lobfile_len*.
CHAR	452	A fixed-length character string. The string length is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 254 bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.
DATE	384	A point in time in accordance with the Gregorian calendar. Each date is a 10-byte character string in International Standards Organization (ISO) format: *yyyy-mm-dd*. The range of the year part is `0001` to `9999`. The range of the month part is `01` to `12`. The range of the day part is `01` to *n*, where *n* depends on the month, using the usual rules for days of the month and leap year. Leading zeros cannot be omitted from any part. IXFCLENG is not used, and should contain blanks. Valid characters within DATE are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.

Table 52. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
DBCLOB	412	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters in the string, and cannot exceed 16 383. The string itself is preceded by a current length indicator, which is a 4-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
DECIMAL	484	A packed decimal number with precision P (as specified by the first three bytes of IXFCLENG in the column descriptor record) and scale S (as specified by the last two bytes of IXFCLENG). The length, in bytes, of a packed decimal number is (P+2)/2. The precision must be an odd number between 1 and 31, inclusive. The packed decimal number is in the internal format specified by IXFTMFRM, where packed decimal for the PC is defined to be the same as packed decimal for the System/370. IXFCSBCP and IXFCDBCP are not significant, and should be zero.
DECFLOAT	996	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 range of a decimal floating-point number is a number with either 16 or 34 digits of precision, and an exponent range of 10-383 to 10+384 or 10-6143 to 10+6144, respectively. The storage length of the 16 digit value is 8 bytes, and the storage length of the 34 digit value is 16 bytes.
FLOATING POINT	480	Either a long (8-byte) or short (4-byte) floating point number, depending on whether IXFCLENG is set to eight or to four. The data is in the internal machine form, as specified by IXFTMFRM. IXFCSBCP and IXFCDBCP are not significant, and should be zero. Four-byte floating point is not supported by the database manager.

Table 52. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
GRAPHIC	468	A fixed-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the number of double-byte characters in the string, and cannot exceed 127. The actual length of the string is twice the value of the IXFCLENG field, in bytes. The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
INTEGER	496	A 4-byte integer in the form specified by IXFTMFRM. It represents a whole number between -2 147 483 648 and +2 147 483 647. IXFCSBCP and IXFCDBCP are not significant, and should be zero. IXFCLENG is not used, and should contain blanks.
LONGVARCHAR	456	A variable-length character string. The maximum length of the string is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 32 767 bytes. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.
LONG VARGRAPHIC	472	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters for the string, and cannot exceed 16 383. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
SMALLINT	500	A 2-byte integer in the form specified by IXFTMFRM. It represents a whole number between -32 768 and +32 767. IXFCSBCP and IXFCDBCP are not significant, and should be zero. IXFCLENG is not used, and should contain blanks.

Table 52. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
TIME	388	A point in time in accordance with the 24-hour clock. Each time is an 8-byte character string in ISO format: *hh.mm.ss*. The range of the hour part is `00` to `24`, and the range of the other parts is `00` to `59`. If the hour is `24`, the other parts are `00`. The smallest time is `00.00.00`, and the largest is `24.00.00`. Leading zeros cannot be omitted from any part. IXFCLENG is not used, and should contain blanks. Valid characters within TIME are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
TIMESTAMP	392	The date and time with microsecond precision. Each time stamp is a character string of the form *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year month day hour minutes seconds microseconds). IXFCLENG is not used, and should contain blanks. Valid characters within TIMESTAMP are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
VARCHAR	448	A variable-length character string. The maximum length of the string, in bytes, is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 254 bytes. The string itself is preceded by a current length indicator, which is a two-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.
VARGRAPHIC	464	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters in the string, and cannot exceed 127. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.

Not all combinations of IXFCSBCP and IXFCDBCP values for PC/IXF character or graphic columns are valid. A PC/IXF character or graphic column with an invalid (IXFCSBCP,IXFCDBCP) combination is an invalid data type.

Table 53. Valid PC/IXF Data Types

PC/IXF Data Type	Valid (IXFCSBCP,IXFCDBCP) Pairs	Invalid (IXFCSBCP,IXFCDBCP) Pairs
CHAR, VARCHAR, or LONG VARCHAR	(0,0), (x,0), or (x,y)	(0,y)
BLOB	(0,0)	(x,0), (0,y), or (x,y)
CLOB	(x,0), (x,y)	(0,0), (0,y)
GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, or DBCLOB	(0,y)	(0,0), (x,0), or (x,y)
Note: x and y are not 0.		

PC/IXF data type descriptions

Table 54. Acceptable Data Type Forms for the PC/IXF File Format

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
BIGINT	A BIGINT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807.
BLOB	A PC/IXF BLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, LONG VARCHAR, BLOB, BLOB_FILE, or BLOB_LOCATION_SPECIFIER column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC BLOB column is also acceptable. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.

Table 54. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
CHAR	A PC/IXF CHAR column is created. The database column length, the SBCS CPGID value, and the DBCS CPGID value are copied to the PC/IXF column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the length of the database column. The data is padded on the right with single-byte spaces (x'20'), if necessary.
CLOB	A PC/IXF CLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, LONG VARCHAR, CLOB, CLOB_FILE, or CLOB_LOCATION_SPECIFIER column is acceptable if the PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.
DATE	A DATE column, identical to the database column, is created.	A PC/IXF column of type DATE is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain dates in a format consistent with the territory code of the target database.
DBCLOB	A PC/IXF DBCLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, DBCLOB, DBCLOB_FILE, or DBCLOB_LOCATION_SPECIFIER column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.

Table 54. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
DECIMAL	A DECIMAL column, identical to the database column, is created. The precision and scale of the column is stored in the column descriptor record.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range of the DECIMAL column into which they are being imported.
DECFLOAT	A DECFLOAT column, identical to the database column, is created. The precision of the column is stored in the column descriptor record.	A column in the following types: SMALLINT, INTEGER, BIGINT (only into DECFLOAT(34)), DECIMAL, FLOAT, REAL, DOUBLE, or DECFLOAT(16) (only into DECFLOAT(34)) is accepted. Other numeric column types are valid for DECFLOAT, but if the value does not fit within the target precision, it is rounded.
FLOAT	A FLOAT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. All values are within range.
GRAPHIC (DBCS only)	A PC/IXF GRAPHIC column is created. The database column length, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the database column length. The data is padded on the right with double-byte spaces (x'8140'), if necessary.
INTEGER	An INTEGER column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -2 147 483 648 to 2 147 483 647.
LONG VARCHAR	A PC/IXF LONG VARCHAR column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.

Table 54. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
LONG VARGRAPHIC (DBCS only)	A PC/IXF LONG VARGRAPHIC column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.
SMALLINT	A SMALLINT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -32 768 to 32 767.
TIME	A TIME column, identical to the database column, is created.	A PC/IXF column of type TIME is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain time data in a format consistent with the territory code of the target database.
TIMESTAMP	A TIMESTAMP column, identical to the database column, is created.	A PC/IXF column of type TIMESTAMP is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain data in the input format for time stamps.
VARCHAR	If the maximum length of the database column is = 254, a PC/IXF VARCHAR column is created. If the maximum length of the database column is > 254, a PC/IXF LONG VARCHAR column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.

Table 54. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
VARGRAPHIC (DBCS only)	If the maximum length of the database column is = 127, a PC/IXF VARGRAPHIC column is created. If the maximum length of the database column is > 127, a PC/IXF LONG VARGRAPHIC column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.

General rules governing PC/IXF file import into databases

The database manager import utility applies the following general rules when importing a PC/IXF file in either an SBCS or a DBCS environment:

- The import utility accepts PC/IXF format files only (IXFHID = 'IXF'). IXF files of other formats cannot be imported.
- The import utility rejects a PC/IXF file with more than 1024 columns.
- When exporting to the IXF format, if identifiers exceed the maximum size supported by the IXF format, the export operation succeeds, but the resulting data file cannot be used by a subsequent import operation using the CREATE mode. SQL27984W is returned.

  Note: The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.
- The value of IXFHSBCP in the PC/IXF H record must equal the SBCS CPGID, or there must be a conversion table between the IXFHSBCP/IXFHDBCP and the SBCS/DBCS CPGID of the target database. The value of IXFHDBCP must equal either '00000', or the DBCS CPGID of the target database. If either of these conditions is not satisfied, the import utility rejects the PC/IXF file, unless the FORCEIN option is specified.
- Invalid data types — new tables

  Import of a PC/IXF file into a *new* table is specified by the CREATE or the REPLACE_CREATE keywords in the IMPORT command. If a PC/IXF column of an invalid data type is selected for import into a new table, the import utility terminates. The entire PC/IXF file is rejected, no table is created, and no data is imported.
- Invalid data types — existing tables

  Import of a PC/IXF file into an *existing* table is specified by the INSERT, the INSERT_UPDATE, the REPLACE or the REPLACE_CREATE keywords in the

IMPORT command. If a PC/IXF column of an invalid data type is selected for import into an existing table, one of two actions is possible:

  – If the target table column is nullable, all values for the invalid PC/IXF column are ignored, and the table column values are set to NULL
  – If the target table column is not nullable, the import utility terminates. The entire PC/IXF file is rejected, and no data is imported. The existing table remains unaltered.

- When importing into a new table, nullable PC/IXF columns generate nullable database columns, and not nullable PC/IXF columns generate not nullable database columns.
- A not nullable PC/IXF column can be imported into a nullable database column.
- A nullable PC/IXF column can be imported into a not nullable database column. If a NULL value is encountered in the PC/IXF column, the import utility rejects the values of all columns in the PC/IXF row that contains the NULL value (the entire row is rejected), and processing continues with the next PC/IXF row. That is, no data is imported from a PC/IXF row that contains a NULL value if a target table column (for the NULL) is not nullable.
- Incompatible Columns — New Table

  If, during import to a *new* database table, a PC/IXF column is selected that is incompatible with the target database column, the import utility terminates. The entire PC/IXF file is rejected, no table is created, and no data is imported.

  Note: IMPORT's FORCEIN option extends the scope of compatible columns.
- Incompatible columns — existing table

  If, during import to an *existing* database table, a PC/IXF column is selected that is incompatible with the target database column, one of two actions is possible:

  – If the target table column is nullable, all values for the PC/IXF column are ignored, and the table column values are set to NULL
  – If the target table column is not nullable, the import utility terminates. The entire PC/IXF file is rejected, and no data is imported. The existing table remains unaltered.

  Note: IMPORT's FORCEIN option extends the scope of compatible columns.
- Invalid values

  If, during import, a PC/IXF column value is encountered that is not valid for the target database column, the import utility rejects the values of all columns in the PC/IXF row that contains the invalid value (the entire row is rejected), and processing continues with the next PC/IXF row.

Data type-specific rules governing PC/IXF file import into databases

- A valid PC/IXF numeric column can be imported into any compatible numeric database column. PC/IXF columns containing 4-byte floating point data are not imported, because this is an invalid data type.
- Database date/time columns can accept values from matching PC/IXF date/time columns (DATE, TIME, and TIMESTAMP), as well as from PC/IXF character columns (CHAR, VARCHAR, and LONG VARCHAR), subject to column length and value compatibility restrictions.
- A valid PC/IXF character column (CHAR, VARCHAR, or LONG VARCHAR) can always be imported into an *existing* database character column marked FOR BIT DATA; otherwise:

  – IXFCSBCP and the SBCS CPGID must agree

– There must be a conversion table for the IXFCSBCP/IXFCDBCP and the SBCS/DBCS
– One set must be all zeros (FOR BIT DATA).

If IXFCSBCP is not zero, the value of IXFCDBCP must equal either zero or the DBCS CPGID of the target database column.

If either of these conditions is not satisfied, the PC/IXF and database columns are incompatible.

When importing a valid PC/IXF character column into a *new* database table, the value of IXFCSBCP must equal either zero or the SBCS CPGID of the database, or there must be a conversion table. If IXFCSBCP is zero, IXFCDBCP must also be zero (otherwise the PC/IXF column is an invalid data type); IMPORT creates a character column marked FOR BIT DATA in the new table. If IXFCSBCP is not zero, and equals the SBCS CPGID of the database, the value of IXFCDBCP must equal either zero or the DBCS CPGID of the database; in this case, the utility creates a character column in the new table with SBCS and DBCS CPGID values equal to those of the database. If these conditions are not satisfied, the PC/IXF and database columns are incompatible.

The FORCEIN option can be used to override code page equality checks. However, a PC/IXF character column with IXFCSBCP equal to zero and IXFCDBCP not equal to zero is an invalid data type, and cannot be imported, even if FORCEIN is specified.

- A valid PC/IXF graphic column (GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC) can always be imported into an *existing* database character column marked FOR BIT DATA, but is incompatible with all other database columns. The FORCEIN option can be used to relax this restriction. However, a PC/IXF graphic column with IXFCSBCP not equal to zero, or IXFCDBCP equal to zero, is an invalid data type, and cannot be imported, even if FORCEIN is specified.

  When importing a valid PC/IXF graphic column into a database graphic column, the value of IXFCDBCP must equal the DBCS CPGID of the target database column (that is, the double-byte code pages of the two columns must agree).
- If, during import of a PC/IXF file into an existing database table, a fixed-length string column (CHAR or GRAPHIC) is selected whose length is greater than the maximum length of the target column, the columns are incompatible.
- If, during import of a PC/IXF file into an existing database table, a variable-length string column (VARCHAR, LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC) is selected whose length is greater than the maximum length of the target column, the columns *are* compatible. Individual values are processed according to the compatibility rules governing the database manager INSERT statement, and PC/IXF values which are too long for the target database column are invalid.
- PC/IXF values imported into a fixed-length database *character* column (that is, a CHAR column) are padded on the right with single-byte spaces (0x20), if necessary, to obtain values whose length equals that of the database column. PC/IXF values imported into a fixed-length database *graphic* column (that is, a GRAPHIC column) are padded on the right with double-byte spaces (0x8140), if necessary, to obtain values whose length equals that of the database column.
- Since PC/IXF VARCHAR columns have a maximum length of 254 bytes, a database VARCHAR column of maximum length *n*, with 254 *n* 4001, must be exported into a PC/IXF LONG VARCHAR column of maximum length *n*.
- Although PC/IXF LONG VARCHAR columns have a maximum length of 32 767 bytes, and database LONG VARCHAR columns have a maximum length

restriction of 32 700 bytes, PC/IXF LONG VARCHAR columns of length greater than 32 700 bytes (but less than 32 768 bytes) are still valid, and can be imported into database LONG VARCHAR columns, but data might be lost.

- Since PC/IXF VARGRAPHIC columns have a maximum length of 127 bytes, a database VARGRAPHIC column of maximum length *n*, with 127 *n* 2001, must be exported into a PC/IXF LONG VARGRAPHIC column of maximum length *n*.
- Although PC/IXF LONG VARGRAPHIC columns have a maximum length of 16 383 bytes, and database LONG VARGRAPHIC columns have a maximum length restriction of 16 350, PC/IXF LONG VARGRAPHIC columns of length greater than 16 350 bytes (but less than 16 384 bytes) are still valid, and can be imported into database LONG VARGRAPHIC columns, but data might be lost.

Table 55 and Table 56 summarize PC/IXF file import into new or existing database tables without the FORCEIN option.

Table 55. Summary of PC/IXF file import without FORCEIN option–numeric types

	DATABASE COLUMN DATA TYPE					
PC/IXF COLUMN DATA TYPE	**SMALL INT**	**INT**	**BIGINT**	**DEC**	**DFP**	**FLT**
-SMALLINT	N					
	E	E	E	E[a]	E	E
-INTEGER		N				
	E[a]	E	E	E[a]	E	E
-BIGINT			N			
	E[a]	E[a]	E	E[a]	E	E
-DECIMAL				N		
	E[a]	E[a]	E[a]	E[a]	E	E
-DECFLOAT						
	E[a]	E[a]	E[a]	E[a]	E	E[a]
-FLOAT						N
	E[a]	E[a]	E[a]	E[a]	E	E

[a] Individual values are rejected if they are out of range for the target numeric data type.

Table 56. Summary of PC/IXF file import without FORCEIN option–character, graphic, and date/time types

	DATABASE COLUMN DATA TYPE						
PC/IXF COLUMN DATA TYPE	**(0,0)**	**(SBCS, 0)[d]**	**(SBCS, DBCS)[b]**	**GRAPH[b]**	**DATE**	**TIME**	**TIME STAMP**
-(0,0)	N						
	E				E[c]	E[c]	E[c]
-(SBCS,0)		N	N				
	E	E	E		E[c]	E[c]	E[c]
-(SBCS, DBCS)			N		E[c]	E[c]	E[c]
	E		E				
-GRAPHIC				N			
	E			E			
-DATE					N		

Table 56. Summary of PC/IXF file import without FORCEIN option–character, graphic, and date/time types (continued)

	DATABASE COLUMN DATA TYPE						
PC/IXF COLUMN DATA TYPE	**(0,0)**	**(SBCS, 0)[d]**	**(SBCS, DBCS)[b]**	**GRAPH[b]**	**DATE**	**TIME**	**TIME STAMP**
					E		
-TIME						N	
						E	
-TIME STAMP							N
							E

[b] Data type is available only in DBCS environments.

[c] Individual values are rejected if they are not valid date or time values.

[d] Data type is not available in DBCS environments.

Note:

1. The table is a matrix of all valid PC/IXF and database manager data types. If a PC/IXF column can be imported into a database column, a letter is displayed in the matrix cell at the intersection of the PC/IXF data type matrix row and the database manager data type matrix column. An 'N' indicates that the utility is creating a new database table (a database column of the indicated data type is created). An 'E' indicates that the utility is importing data to an existing database table (a database column of the indicated data type is a valid target).
2. Character string data types are distinguished by code page attributes. These attributes are shown as an ordered pair (SBCS,DBCS), where:
   - SBCS is either zero or denotes a non-zero value of the single-byte code page attribute of the character data type
   - DBCS is either zero or denotes a non-zero value of the double-byte code page attribute of the character data type.
3. If the table indicates that a PC/IXF character column can be imported into a database character column, the values of their respective code page attribute pairs satisfy the rules governing code page equality.

Differences between PC/IXF and Version 0 System/370 IXF

The following describes differences between PC/IXF, used by the database manager, and Version 0 System/370 IXF, used by several host database products:

- PC/IXF files are ASCII, rather than EBCDIC oriented. PC/IXF files have significantly expanded code page identification, including new code page identifiers in the H record, and the use of actual code page values in the column descriptor records. There is also a mechanism for marking columns of character data as FOR BIT DATA. FOR BIT DATA columns are of special significance, because transforms which convert a PC/IXF file format to or from any other IXF or database file format cannot perform any code page translation on the values contained in FOR BIT DATA columns.
- Only the machine data form is permitted; that is, the IXFTFORM field must always contain the value `M`. Furthermore, the machine data must be in PC forms; that is, the IXFTMFRM field must contain the value `PC`. This means that integers, floating point numbers, and decimal numbers in data portions of PC/IXF data records must be in PC forms.

- Application (A) records are permitted anywhere after the H record in a PC/IXF file. They are not counted when the value of the IXFHHCNT field is computed.
- Every PC/IXF record begins with a record length indicator. This is a 6-byte character representation of an integer value containing the length, in bytes, of the PC/IXF record not including the record length indicator itself; that is, the total record length minus 6 bytes. The purpose of the record length field is to enable PC programs to identify record boundaries.
- To facilitate the compact storage of variable-length data, and to avoid complex processing when a field is split into multiple records, PC/IXF does not support Version 0 IXF X records, but does support D record identifiers. Whenever a variable-length field or a nullable field is the last field in a data D record, it is not necessary to write the entire maximum length of the field to the PC/IXF file.

FORCEIN option

The `forcein` file type modifier permits import of a PC/IXF file despite code page differences between data in the PC/IXF file and the target database. It offers additional flexibility in the definition of compatible columns.

General semantics of forcein

The following general semantics apply when using the `forcein` file type modifier in either an SBCS or a DBCS environment:

- The `forcein` file type modifier should be used with caution. It is usually advisable to attempt an import without this option enabled. However, because of the generic nature of the PC/IXF data interchange architecture, some PC/IXF files might contain data types or values that cannot be imported without intervention.
- Import with `forcein` to a *new* table might yield a different result than import to an existing table. An existing table has predefined target data types for each PC/IXF data type.
- When LOB data is exported with the `lobsinfile` file type modifier, and the files move to another client with a different code page, then, unlike other data, the CLOBS and DBCLOBS in the separate files are not converted to the client code page when imported or loaded into a database.

Code page semantics for forcein

The following code page semantics apply when using the `forcein` file type modifier in either an SBCS or a DBCS environment:

- The `forcein` file type modifier disables all import utility code page comparisons.

  This rule applies to code page comparisons at the column level and at the file level as well, when importing to a new or an existing database table. At the column (for example, data type) level, this rule applies only to the following database manager and PC/IXF data types: character (CHAR, VARCHAR, and LONG VARCHAR), and graphic (GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC). The restriction follows from the fact that code page attributes of other data types are not relevant to the interpretation of data type values.
- `forcein` does not disable inspection of code page attributes to determine data types.

  For example, the database manager allows a CHAR column to be declared with the FOR BIT DATA attribute. Such a declaration sets both the SBCS CPGID and the DBCS CPGID of the column to zero; it is the zero value of these CPGIDs that identifies the column values as bit strings (rather than character strings).
- `forcein` does not imply code page translation.

Values of data types that are sensitive to the forcein file type modifier are copied "as is". No code point mappings are employed to account for a change of code page environments. Padding of the imported value with spaces might be necessary in the case of fixed length target columns.

- When data is imported to an *existing* table using forcein:
  - The code page value of the target database table and columns always prevails.
  - The code page value of the PC/IXF file and columns is ignored.

  This rule applies whether or not forcein is used. The database manager does not permit changes to a database or a column code page value once a database is created.
- When importing to a *new* table using forcein:
  - The code page value of the target database prevails.
  - PC/IXF character columns with IXFCSBCP = IXFCDBCP = 0 generate table columns marked FOR BIT DATA.
  - All other PC/IXF character columns generate table character columns with SBCS and DBCS CPGID values equal to those of the database.
  - PC/IXF graphic columns generate table graphic columns with an SBCS CPGID of "undefined", and a DBCS CPGID equal to that of the database (DBCS environment only).

Example of forcein

Consider a PC/IXF CHAR column with IXFCSBCP = '00897' and IXFCDBCP = '00301'. This column is to be imported into a database CHAR column whose SBCS CPGID = '00850' and DBCS CPGID = '00000'. Without forcein, the utility terminates, and no data is imported, or the PC/IXF column values are ignored, and the database column contains NULLs (if the database column is nullable). With forcein, the utility proceeds, ignoring code page incompatibilities. If there are no other data type incompatibilities (such as length, for example), the values of the PC/IXF column are imported "as is", and become available for interpretation under the database column code page environment.

The following two tables show:

- The code page attributes of a column created in a *new* database table when a PC/IXF file data type with specified code page attributes is imported.
- That the import utility rejects PC/IXF data types if they are invalid or incompatible.

Table 57. Summary of Import Utility Code Page Semantics (New Table) for SBCS. This table assumes there is no conversion table between a and x. If there were, items 3 and 4 would work successfully without forcein.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN	
	Without forcein	With forcein
(0,0)	(0,0)	(0,0)
(a,0)	(a,0)	(a,0)
(x,0)	reject	(a,0)
(x,y)	reject	(a,0)
(a,y)	reject	(a,0)
(0,y)	reject	(0,0)

Table 57. Summary of Import Utility Code Page Semantics (New Table) for SBCS (continued). This table assumes there is no conversion table between a and x. If there were, items 3 and 4 would work successfully without `forcein`.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN	
	Without `forcein`	**With `forcein`**
Note: 1. See the notes for Table 58.		

Table 58. Summary of Import Utility Code Page Semantics (New Table) for DBCS. This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN	
	Without `forcein`	**With `forcein`**
(0,0)	(0,0)	(0,0)
(a,0)	(a,b)	(a,b)
(x,0)	reject	(a,b)
(a,b)	(a,b)	(a,b)
(x,y)	reject	(a,b)
(a,y)	reject	(a,b)
(x,b)	reject	(a,b)
(0,b)	(-,b)	(-,b)
(0,y)	reject	(-,b)

Note:

1. Code page attributes of a PC/IXF data type are shown as an ordered pair, where x represents a non-zero single-byte code page value, and y represents a non-zero double-byte code page value. A '-' represents an undefined code page value.
2. The use of different letters in various code page attribute pairs is deliberate. Different letters imply different values. For example, if a PC/IXF data type is shown as (x,y), and the database column as (a,y), x does not equal a, but the PC/IXF file and the database have the same double-byte code page value y.
3. Only character and graphic data types are affected by the `forcein` code page semantics.
4. It is assumed that the database containing the new table has code page attributes of (a,0); therefore, all character columns in the new table must have code page attributes of either (0,0) or (a,0).

   In a DBCS environment, it is assumed that the database containing the new table has code page attributes of (a,b); therefore, all graphic columns in the new table must have code page attributes of (-,b), and all character columns must have code page attributes of (a,b). The SBCS CPGID is shown as '-', because it is undefined for graphic data types.
5. The data type of the result is determined by the rules described in Data type semantics for `forcein`.
6. The `reject` result is a reflection of the rules for invalid or incompatible data types.

The following two tables show:

- That the import utility accepts PC/IXF data types with various code page attributes into an *existing* table column (the *target* column) having the specified code page attributes.

- That the import utility does not permit a PC/IXF data type with certain code page attributes to be imported into an *existing* table column having the code page attributes shown. The utility rejects PC/IXF data types if they are invalid or incompatible.

Table 59. Summary of Import Utility Code Page Semantics (Existing Table) for SBCS. This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without `forcein`	With `forcein`
(0,0)	(0,0)	accept	accept
(a,0)	(0,0)	accept	accept
(x,0)	(0,0)	accept	accept
(x,y)	(0,0)	accept	accept
(a,y)	(0,0)	accept	accept
(0,y)	(0,0)	accept	accept
(0,0)	(a,0)	null or reject	accept
(a,0)	(a,0)	accept	accept
(x,0)	(a,0)	null or reject	accept
(x,y)	(a,0)	null or reject	accept
(a,y)	(a,0)	null or reject	accept
(0,y)	(a,0)	null or reject	null or reject

Note:

1. See the notes for Table 57 on page 417.
2. The `null or reject` result is a reflection of the rules for invalid or incompatible data types.

Table 60. Summary of Import Utility Code Page Semantics (Existing Table) for DBCS. This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without `forcein`	With `forcein`
(0,0)	(0,0)	accept	accept
(a,0)	(0,0)	accept	accept
(x,0)	(0,0)	accept	accept
(a,b)	(0,0)	accept	accept
(x,y)	(0,0)	accept	accept
(a,y)	(0,0)	accept	accept
(x,b)	(0,0)	accept	accept
(0,b)	(0,0)	accept	accept
(0,y)	(0,0)	accept	accept

Table 60. Summary of Import Utility Code Page Semantics (Existing Table) for DBCS (continued). This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without `forcein`	With `forcein`
(0,0)	(a,b)	null or reject	accept
(a,0)	(a,b)	accept	accept
(x,0)	(a,b)	null or reject	accept
(a,b)	(a,b)	accept	accept
(x,y)	(a,b)	null or reject	accept
(a,y)	(a,b)	null or reject	accept
(x,b)	(a,b)	null or reject	accept
(0,b)	(a,b)	null or reject	null or reject
(0,y)	(a,b)	null or reject	null or reject
(0,0)	(-,b)	null or reject	accept
(a,0)	(-,b)	null or reject	null or reject
(x,0)	(-,b)	null or reject	null or reject
(a,b)	(-,b)	null or reject	null or reject
(x,y)	(-,b)	null or reject	null or reject
(a,y)	(-,b)	null or reject	null or reject
(x,b)	(-,b)	null or reject	null or reject
(0,b)	(-,b)	accept	accept
(0,y)	(-,b)	null or reject	accept

Note:

1. See the notes for Table 57 on page 417.
2. The `null` or `reject` result is a reflection of the rules for invalid or incompatible data types.

Data type semantics for forcein

The `forcein` file type modifier permits import of certain PC/IXF columns into target database columns of unequal and otherwise incompatible data types. The following data type semantics apply when using `forcein` in either an SBCS or a DBCS environment (except where noted):

- In SBCS environments, `forcein` permits import of:
  - A PC/IXF BIT data type (IXFCSBCP = 0 = IXFCDBCP for a PC/IXF character column) into a database character column (non-zero SBCS CPGID, and DBCS CPGID = 0); existing tables only
  - A PC/IXF MIXED data type (non-zero IXFCSBCP and IXFCDBCP) into a database character column; both new and existing tables
  - A PC/IXF GRAPHIC data type into a database FOR BIT DATA column (SBCS CPGID = 0 = DBCS CPGID); new tables only (this is always permitted for existing tables).

- The `forcein` file type modifier does not extend the scope of valid PC/IXF data types.

  PC/IXF columns with data types not defined as valid PC/IXF data types are invalid for import with or without `forcein`.
- In DBCS environments, `forcein` permits import of:
  - A PC/IXF BIT data type into a database character column
  - A PC/IXF BIT data type into a database graphic column; however, if the PC/IXF BIT column is of fixed length, that length must be even. A fixed length PC/IXF BIT column of odd length is not compatible with a database graphic column. A varying-length PC/IXF BIT column *is* compatible whether its length is odd or even, although an odd-length value from a varying-length column is an invalid value for import into a database graphic column
  - A PC/IXF MIXED data type into a database character column.

Table 61 summarizes PC/IXF file import into new or existing database tables with `forcein` specified.

Table 61. Summary of PC/IXF File Import with forcein

	DATABASE COLUMN DATA TYPE											
PC/IXF COLUMN DATA TYPE	**SMALL INT**	**INT**	**BIGINT**	**DEC**	**FLT**	**(0,0)**	**(SBCS, 0)[e]**	**(SBCS, DBCS)[b]**	**GRAPH[b]**	**DATE**	**TIME**	**TIME STAMP**
-SMALLINT	N											
	E	E	E	E[a]	E							
-INTEGER		N										
	E[a]	E	E	E[a]	E							
-BIGINT			N									
	E[a]	E[a]	E	E[a]	E							
-DECIMAL				N								
	E[a]	E[a]	E[a]	E[a]	E							
-FLOAT					N							
	E[a]	E[a]	E[a]	E[a]	E							
-(0,0)						N						
						E	E w/F	E w/F	E w/F	E[c]	E[c]	E[c]
-(SBCS,0)							N	N				
						E	E	E		E[c]	E[c]	E[c]
-(SBCS, DBCS)							N w/F[d]	N		E[c]	E[c]	E[c]
						E	E w/F	E				
-GRAPHIC						N w/F[d]			N			
						E			E			
-DATE										N		
										E		

Table 61. Summary of PC/IXF File Import with forcein (continued)

	DATABASE COLUMN DATA TYPE											
PC/IXF COLUMN DATA TYPE	**SMALL INT**	**INT**	**BIGINT**	**DEC**	**FLT**	**(0,0)**	**(SBCS, 0)[e]**	**(SBCS, DBCS)[b]**	**GRAPH[b]**	**DATE**	**TIME**	**TIME STAMP**
-TIME											N	
											E	
-TIME STAMP												N
												E

Note: If a PC/IXF column can be imported into a database column only with `forcein`, the string 'w/F' is displayed together with an 'N' or an 'E'. An 'N' indicates that the utility is creating a new database table; an 'E' indicates that the utility is importing data to an existing database table. The `forcein` file type modifier affects compatibility of character and graphic data types only.

[a] Individual values are rejected if they are out of range for the target numeric data type.

[b] Data type is available only in DBCS environments.

[c] Individual values are rejected if they are not valid date or time values.

[d] Applies only if the source PC/IXF data type is not supported by the target database.

[e] Data type is not available in DBCS environments.

Worksheet File Format (WSF)

Lotus 1-2-3 and Symphony products use the same basic format, with additional functions added at each new release. The database manager supports the subset of the worksheet records that are the same for all the Lotus products. That is, for the releases of Lotus 1-2-3 and Symphony products supported by the database manager, all file names with any three-character extension are accepted; for example: WKS, WK1, WRK, WR1, WJ2.

Each WSF file represents one worksheet. The database manager uses the following conventions to interpret worksheets and to provide consistency in worksheets generated by its export operations:

- Cells in the first row (ROW value 0) are reserved for descriptive information about the entire worksheet. All data within this row is optional. It is ignored during import.
- Cells in the second row (ROW value 1) are used for column labels.
- The remaining rows are data rows (records, or rows of data from the table).
- Cell values under any column heading are values for that particular column or field.
- A NULL value is indicated by the absence of a real cell content record (for example, no integer, number, label, or formula record) for a particular column within a row of cell content records.

  Note: A row of NULLs will be neither imported nor exported.

To create a file that is compliant with the WSF format during an export operation, some loss of data might occur.

WSF files use a Lotus code point mapping that is not necessarily the same as existing code pages supported by DB2 database. As a result, when importing or exporting a WSF file, data is converted from the Lotus code points to or from the code points used by the application code page. DB2 supports conversion between the Lotus code points and code points defined by code pages 437, 819, 850, 860, 863, and 865.

Note: For multi-byte character set users, no conversions are performed.

Unicode considerations for data movement

The export, import, and load utilities are not supported when they are used with a Unicode client connected to a non-Unicode database.

The DEL, ASC, and PC/IXF file formats are supported for a Unicode database, as described in this section. The WSF format is not supported.

When exporting from a Unicode database to an ASCII delimited (DEL) file, all character data is converted to the application code page. Both character string and graphic string data are converted to the same SBCS or MBCS code page of the client. This is expected behavior for the export of any database, and cannot be changed, because the entire delimited ASCII file can have only one code page. Therefore, if you export to a delimited ASCII file, only those UCS-2 characters that exist in your application code page will be saved. Other characters are replaced with the default substitution character for the application code page. For UTF-8 clients (code page 1208), there is no data loss, because all UCS-2 characters are supported by UTF-8 clients.

When importing from an ASCII file (DEL or ASC) to a Unicode database, character string data is converted from the application code page to UTF-8, and graphic string data is converted from the application code page to UCS-2. There is no data loss. If you want to import ASCII data that has been saved under a different code page, you should change the data file code page before issuing the IMPORT command. You can specify the code page of the data file by setting the **DB2CODEPAGE** registry variable to the code page of the ASCII data file or by using the `codepage` file type modifier.

The range of valid ASCII delimiters for SBCS and MBCS clients is identical to what is currently supported by IBM DB2 V9.1 for those clients. The range of valid delimiters for UTF-8 clients is X'01' to X'7F', with the usual restrictions.

When exporting from a Unicode database to a PC/IXF file, character string data is converted to the SBCS/MBCS code page of the client. Graphic string data is not converted, and is stored in UCS-2 (code page 1200). There is no data loss.

When importing from a PC/IXF file to a Unicode database, character string data is assumed to be in the SBCS/MBCS code page stored in the PC/IXF header, and graphic string data is assumed to be in the DBCS code page stored in the PC/IXF header. Character string data is converted by the import utility from the code page specified in the PC/IXF header to the code page of the client, and then from the client code page to UTF-8 (by the INSERT statement). Graphic string data is converted by the import utility from the DBCS code page specified in the PC/IXF header directly to UCS-2 (code page 1200).

The load utility places the data directly into the database and, by default, assumes data in ASC or DEL files to be in the code page of the database. Therefore, by default, no code page conversion takes place for ASCII files. When the code page for the data file has been explicitly specified (using the `codepage` modifier), the load utility uses this information to convert from the specified code page to the database code page before loading the data. For PC/IXF files, the load utility always converts from the code pages specified in the IXF header to the database code page (1208 for CHAR, and 1200 for GRAPHIC).

The code page for DBCLOB files is always 1200 for UCS-2. The code page for CLOB files is the same as the code page for the data files being imported, loaded or exported. For example, when loading or importing data using the PC/IXF format, the CLOB file is assumed to be in the code page specified by the PC/IXF header. If the DBCLOB file is in ASC or DEL format, the load utility assumes that CLOB data is in the code page of the database, while the import utility assumes it to be in the code page of the client application.

The `nochecklengths` modifier is always specified for a Unicode database, because:

- Any SBCS can be connected to a database for which there is no DBCS code page
- Character strings in UTF-8 format usually have different lengths than those in client code pages.

Considerations for code pages 1394, 1392, and 5488

The import, export and load utilities can be used to transfer data from the Chinese code page GB 18030 (code page identifier 1392 and 5488) and the Japanese code page ShiftJISX 0213 (code page identifier 1394) to DB2 Unicode databases. In addition, the export utility can be used to transfer data from DB2 Unicode databases to GB 18030 or ShiftJIS X0213 code page data.

For example, the following command will load the Shift_JISX0213 data file `u/jp/user/x0213/data.del` residing on a remotely connected client into MYTABLE:

```
db2 load client from /u/jp/user/x0213/data.del
of del modified by codepage=1394 insert into mytable
```

where MYTABLE is located on a DB2 Unicode database.

Since only connections between a Unicode client and a Unicode server are supported, you need to use either a Unicode client or set the DB2 registry variable **DB2CODEPAGE** to 1208 prior to using the load, import, or export utilities.

Conversion from code page 1394, 1392, or 5488 to Unicode can result in expansion. For example, a 2-byte character can be stored as two 16-bit Unicode characters in the GRAPHIC columns. You need to ensure the target columns in the Unicode database are wide enough to contain any expanded Unicode byte.

Incompatibilities

For applications connected to a Unicode database, graphic string data is always in UCS-2 (code page 1200). For applications connected to non-Unicode databases, the graphic string data is in the DBCS code page of the application, or not allowed if the application code page is SBCS. For example, when a 932 client is connected to a Japanese non-Unicode database, the graphic string data is in code page 301. For the 932 client applications connected to a Unicode database, the graphic string data is in UCS-2 encoding.

Character set and national language support

The DB2 data movement utilities offer the following national language support (NLS):

- The import and the export utilities provide automatic code page conversion from a client code page to the server code page.
- For the load utility, data can be converted from any code page to the server code page by using the `codepage` modifier with DEL and ASC files.
- For all utilities, IXF data is automatically converted from its original code page (as stored in the IXF file) to the server code page.

Unequal code page situations, involving expansion or contraction of the character data, can sometimes occur. For example, Japanese or Traditional-Chinese Extended UNIX Code (EUC) and double-byte character sets (DBCS) might encode different lengths for the same character. Normally, comparison of input data length to target column length is performed before reading in any data. If the input length is greater than the target length, NULLs are inserted into that column if it is nullable. Otherwise, the request is rejected. If the `nochecklengths` file type modifier is specified, no initial comparison is performed, and an attempt is made to import or load the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is imported or loaded.

XML data movement

Support for XML data movement is provided by the load, import and export utilities.

Importing XML data

The import utility can be used to insert XML documents into a regular relational table. Only well-formed XML documents can be imported.

Use the XML FROM option of the IMPORT command to specify the location of the XML documents to import. The XMLVALIDATE option specifies how imported documents should be validated. You can select to have the imported XML data validated against a schema specified with the IMPORT command, against a schema identified by a schema location hint inside of the source XML document, or by the schema identified by the XML Data Specifier in the main data file. You can also use the XMLPARSE option to specify how whitespace is handled when the XML document is imported. The `xmlchar` and `xmlgraphic` file type modifiers allow you to specify the encoding characteristics for the imported XML data.

Loading XML data

The load utility offers an efficient way to insert large volumes of XML data into a table. This utility also allows certain options unavailable with the import utility, such as the ability to load from a user-defined cursor.

Like the IMPORT command, with the LOAD command you can specify the location of the XML data to load, validation options for the XML data, and how whitespace is handled. As with IMPORT, you can use the `xmlchar` and `xmlgraphic` file type modifiers to specify the encoding characteristics for the loaded XML data.

Exporting XML data

Data may be exported from tables that include one or more columns with an XML data type. Exported XML data is stored in files separate from the main data file containing the exported relational data. Information about each exported XML document is represented in the main exported data file by an XML data specifier (XDS). The XDS is a string that specifies the name of the system file in which the XML document is stored, the exact location and length of the XML document inside of this file, and the XML schema used to validate the XML document.

You can use the XMLFILE, XML TO, and XMLSAVESCHEMA parameters of the EXPORT command to specify details about how exported XML documents are stored. The `xmlinsepfiles`, `xmlnodeclaration`, `xmlchar`, and `xmlgraphic` file type modifiers allow you to specify further details about the storage location and the encoding of the exported XML data.

Important considerations for XML data movement

There are a number of considerations to keep in mind when importing or exporting XML data:

- Exported XML data is always stored separately from the main data file containing exported relational data.
- By default, the export utility writes XML data in Unicode. You can use the `xmlchar` file type modifier to have XML data written in the character code page. The `xmlgraphic` file type modifier specifies that XML data is written in the graphic code page, which is UTF-16 regardless of the application code page.
- Beginning with version 9.5, XML data can be stored in non-Unicode databases, as the data will be converted from the database codepage to UTF-8 before insertion. In order to avoid the possible introduction of substitution characters during XML parsing, character data to be inserted should consist only of codepoints that are part of the database codepage. Setting the `enable_xmlchar` configuration parameter to `no` will block the insertion of character data types during XML parsing, restricting insertion to data types that do not undergo codepage conversion, such as BIT DATA, BLOB, or pureXML™.
- For the import and load utilities, unless the XML document to import contains a declaration tag that includes an encoding attribute, this document is assumed to be in Unicode. You can use the `xmlchar` file type modifier to indicate that XML documents to import are encoded in the character code page, while the `xmlgraphic` file type modifier indicates that XML documents to import are encoded in UTF-16.
- For the import and load utilities, rows which contain documents that are not well-formed will be rejected.
- If the XMLVALIDATE option is specified for the import utility or the load utility, documents which successfully validate against their matching schema will be annotated with the schema information as they are inserted into a table. Rows containing documents that fail to validate against their matching schema will be rejected.
- You can use the export utility with an XQuery specification to export Query and XPath Data Model (XDM) instances that are not well-formed XML documents. However, exported XML documents that are not well-formed cannot be imported directly into an XML column, since columns defined with the XML data type can contain only complete XML documents.
- **CPU_PARALLELISM** during a load is reduced to 1 if statistics are being collected.

- An XML load operation requires the use of shared sort memory to proceed. Thus, you need to enable **SHEAPTHRES_SHR** or **INTRA_PARALLEL**, or turn on the connection concentrator. Note that by default, **SHEAPTHRES_SHR** is set to a value, so shared sort memory is available on the default configuration.
- Load operations into a table containing an XML column cannot have the SOURCEUSEREXIT option, the **SAVECOUNT** parameter, or the `anyorder` file type modifier specified.
- As with LOB files, XML files have to reside on the server side.

LOB and XML file behavior when importing and exporting

LOB and XML files share certain behaviors and compatibilities that can be used when importing and exporting data.

Export When exporting data, if one or more LOB paths are specified with the LOBS TO option, the export utility will cycle between the paths to write each successive LOB value to the appropriate LOB file. Similarly, if one or more XML paths are specified with the XML TO option, the export utility will cycle between the paths to write each successive XQuery and XPath Data Model (XDM) instance to the appropriate XML file. By default, LOB values and XDM instances are written to the same path to which the exported relational data is written. Unless the LOBSINSEPFILES or XMLINSEPFILES file type modifier is set, both LOB files and XML files can have multiple values concatenated to the same file.

The LOBFILE option provides a means to specify the base name of the LOB files generated by the export utility. Similarly, the XMLFILE option provides a means to specify the base name of the XML files generated by the export utility. The default LOB file base name is the name of the exported data file, with the extension `.lob`. The default XML file base name is the name of the exported data file, with the extension `.xml`. The full name of the exported LOB file or XML file therefore consists of the base name, followed by a number extension that is padded to three digits, and the extension `.lob` or `.xml`.

Import

When importing data, a LOB Location Specifier (LLS) is compatible with an XML target column, and an XML Data Specifier (XDS) is compatible with a LOB target column. If the LOBS FROM option is not specified, the LOB files to import are assumed to reside in the same path as the input relational data file. Similarly, if the XML FROM option is not specified, the XML files to import are assumed to reside in the same path as the input relational data file.

Export examples

In the following example, all LOB values are written to the file `/mypath/t1export.del.001.lob`, and all XDM instances are written to the file `/mypath/t1export.del.001.xml`:

```
EXPORT TO /mypath/t1export.del OF DEL MODIFIED BY LOBSINFILE
SELECT * FROM USER.T1
```

In the following example, the first LOB value is written to the file `/lob1/t1export.del.001.lob`, the second is written to the file `/lob2/t1export.del.002.lob`, the third is appended to `/lob1/t1export.del.001.lob`, the fourth is appended to `/lob2/t1export.del.002.lob`, and so on:

```
EXPORT TO /mypath/t1export.del OF DEL LOBS TO /lob1,/lob2
MODIFIED BY LOBSINFILE SELECT * FROM USER.T1
```

In the following example, the first XDM instance is written to the file `/xml1/xmlbase.001.xml`, the second is written to the file `/xml2/xmlbase.002.xml`, the third is written to `/xml1/xmlbase.003.xml`, the fourth is written to `/xml2/xmlbase.004.xml`, and so on:

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /xml1,/xml2 XMLFILE xmlbase
   MODIFIED BY XMLINSEPFILES SELECT * FROM USER.T1
```

Import examples

For a table "mytable" that contains a single XML column, and the following IMPORT command:

```
IMPORT FROM myfile.del of del LOBS FROM /lobpath XML FROM /xmlpath
MODIFIED BY LOBSINFILE XMLCHAR replace into mytable
```

If "myfile.del" contains the following data:

```
mylobfile.001.lob.123.456/
```

The import utility will try to import an XML document from the file `/lobpath/mylobfile.001.lob`, starting at file offset 123, with its length being 456 bytes.

The file "mylobfile.001.lob" is assumed to be in the LOB path, as opposed to the XML path, since the value is referred to by a LOB Location Specifier (LLS) instead of an XML Data Specifier (XDS).

The document is assumed to be encoded in the character codepage, since the XMLCHAR file type modifier is specified.

XML data specifier

XML data moved with the export, import and load utilities must be stored in files separate from the main data file. The XML data is represented in the main data file with an XML data specifier (XDS).

The XDS is a string represented as an XML tag named "XDS", which has attributes that describe information about the actual XML data in the column; such information includes the name of the file that contains the actual XML data, and the offset and length of the XML data within that file. The attributes of the XDS are described below.

FIL The name of the file that contains the XML data.

OFF The byte offset of the XML data in the file named by the FIL attribute, where the offset begins from 0.

LEN The length in bytes of the XML data in the file named by the FIL attribute.

SCH The fully qualified SQL identifier of the XML schema that is used to validate this XML document. The schema and name components of the SQL identifier are stored as the "OBJECTSCHEMA" and "OBJECTNAME" values, respectively, of the row in the SYSCAT.XSROBJECTS catalog table that corresponds to this XML schema.

The XDS is interpreted as a character field in the data file and is subject to the parsing behavior for character columns of the file format. For the delimited ASCII file format (DEL), for example, if the character delimiter is present in the XDS, it

must be doubled. The special characters <, >, &, ′, ″ within the attribute values must always be escaped. Case-sensitive object names must be placed between " character entities.

Examples

Consider a FIL attribute with the value `abc&"def".del`. To include this XDS in a delimited ASCII file, where the character delimiter is the ″ character, the ″ characters are doubled and special characters are escaped.

```
<XDS FIL=""abc&amp;&quot;def&quot;.del"" />
```

The following example shows an XDS as it would appear in a delimited ASCII data file. XML data is stored in the file `xmldocs.xml.001` beginning at byte offset 100 with a length of 300 bytes. Because this XDS is within an ASCII file delimited with double quotation marks, the double quotation marks within the XDS tag itself must be doubled.

```
"<XDS FIL = ""xmldocs.xml.001"" OFF=""100"" LEN=""300"" />"
```

The following example shows the fully qualified SQL identifier `ANTHONY.purchaseOrderTest`. The case-sensitive portion of the identifier must be placed between " character entities in the XDS:

```
"<XDS FIL='/home/db2inst1/xmlload/a.xml' OFF='0' LEN='6758'
   SCH='ANTHONY.&quot;purchaseOrderTest&quot;' />"
```

Query and XPath Data Model

XML data can be accessed in a database table either by use of the XQuery functions available in SQL, or by invoking XQuery directly. An instance of the Query and XPath Data Model (XDM) can be a well-formed XML document, a sequence of nodes, a sequence of atomic values, or any combination of nodes and atomic values.

Individual XDM instances can be written to one or more XML files by means of the EXPORT command.

Part 2. Appendixes

© Copyright IBM Corp. 1993, 2008

Appendix A. Differences between the import and load utility

The following table summarizes the important differences between the DB2 load and import utilities.

Import Utility	Load Utility
Slow when moving large amounts of data.	Faster than the import utility when moving large amounts of data, because the load utility writes formatted pages directly into the database.
Limited exploitation of intra-partition parallelism. Intra-partition parallelism can only be achieved through concurrent invocations of the import utility in ALLOW WRITE ACCESS mode.	Exploitation of intra-partition parallelism. Typically, this requires symmetric multiprocessor (SMP) machines.
No FASTPARSE support.	FASTPARSE support, providing reduced data checking of user-supplied data.
Supports hierarchical data.	Does not support hierarchical data.
Creation of tables, hierarchies, and indexes supported with PC/IXF format.	Tables and indexes must exist.
No support for importing into materialized query tables.	Support for loading into materialized query tables.
WSF format is supported.	WSF format is not supported.
No BINARYNUMERICS support.	BINARYNUMERICS support.
No PACKEDDECIMAL support.	PACKEDDECIMAL support.
No ZONEDDECIMAL support.	ZONEDDECIMAL support.
Cannot override columns defined as GENERATED ALWAYS.	Can override GENERATED ALWAYS columns, by using the `generatedoverride` and `identityoverride` file type modifiers.
Supports import into tables, views and nicknames.	Supports loading into tables only.
All rows are logged.	Minimal logging is performed.
Trigger support.	No trigger support.
If an import operation is interrupted, and a *commitcount* was specified, the table is usable and will contain the rows that were loaded up to the last COMMIT. The user can restart the import operation, or accept the table as is.	If a load operation is interrupted, and a *savecount* was specified, the table remains in Load Pending state and cannot be used until the load operation is restarted, a load terminate operation is invoked, or until the table space is restored from a backup image created some time before the attempted load operation.
Space required is approximately equivalent to the size of the largest index plus 10%. This space is obtained from the temporary table spaces within the database.	Space required is approximately equivalent to the sum of the size of all indexes defined on the table, and can be as much as twice this size. This space is obtained from temporary space within the database.
All constraints are validated during an import operation.	The load utility checks for uniqueness and computes generated column values, but all other constraints must be checked using SET INTEGRITY.

© IBM Corporation 1993, 2008

Import Utility	Load Utility
The key values are inserted into the index one at a time during an import operation.	The key values are sorted and the index is built after the data has been loaded.
If updated statistics are required, the runstats utility must be run after an import operation.	Statistics can be gathered during the load operation if all the data in the table is being replaced.
You can import into a host database through DB2 Connect.	You cannot load into a host database.
Import files must exist on the client from which the import utility is invoked.	Depending on the options specified, load files or pipes can reside either on the database partition(s) that contain the database, or on the remotely connected client from which the load utility is invoked. **Note:** LOBs and XML data can only be read from the server side.
A backup image is not required. Because the import utility uses SQL inserts, the activity is logged, and no backups are required to recover these operations in case of failure.	A backup image can be created during the load operation.

Appendix B. Bind files used by the export, import, and load utilities

The following table lists bind files with their default isolation levels, as well as which utilities use them and for what purpose.

Bind File (Default Isolation Level)	Utility/Purpose
db2ueiwi.bnd (CS)	Import/Export. Used to query information about table columns and indexes.
db2uexpm.bnd (CS)	Export. Used to fetch from the query specified for the export operation.
db2uimpm.bnd (RS)	Import. Used to insert data from the source data file into the target table when INSERT, REPLACE or REPLACE_CREATE option is used. **Note:** The CREATE and REPLACE_CREATE options of the IMPORT command are deprecated and might be removed in a future release.
db2uipkg.bnd (CS)	Import. Used to check bind options.
db2uiici.bnd (RR)	Import. Used to create indexes when the IXF CREATE option is specified.
db2ucktb.bnd (CS)	Load. Used to perform general initialization processes for a load operation.
db2ulxld.bnd (CS)	Load. Used to process the query provided during a load from cursor operation.
db2uigsi.bnd (RS on UNIX based systems, RR on all other platforms)	Import/Export. Used to drop indexes and check for referential constraints for an import replace operation. Also used to retrieve identity column information for exporting IXF files.
db2uiict.bnd (RR)	Import. Used to create tables when the IXF CREATE option is specified.
db2uqtpd.bnd (RR)	Import/Export. Used to perform processing for hierarchical tables.
db2uqtnm.bnd (RR)	Import. Used to perform processing for hierarchical tables when the IXF CREATE option is specified.
db2uimtb.bnd (RS)	Import. Used to perform general initialization processes for an import operation.
db2uImpInsUpdate.bnd (RS)	Import. Used to insert data from the source data file into the target table when INSERT_UPDATE option is used. Cannot be bound with the INSERT BUF option.

© Copyright IBM Corp. 1993, 2008

Appendix C. How to read the syntax diagrams

Throughout this book, syntax is described using the structure defined as follows:

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

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

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

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

The —►◄ symbol indicates the end of a syntax diagram.

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

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

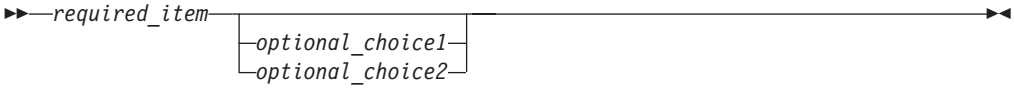
Optional items appear below the main path.

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

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

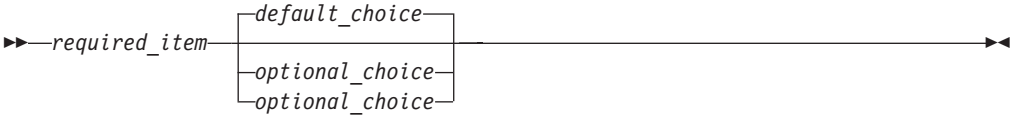
If you *must* choose one of the items, one item of the stack appears on the main path.

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



© Copyright IBM Corp. 1993, 2008

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



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

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

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

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

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

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

parameter-block:

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

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

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

Appendix D. Collecting data for data movement problems

If you are experiencing problems while performing data movement commands and you cannot determine the cause of the problem, collect diagnostic data that either you or IBM Software Support can use to diagnose and resolve the problem.

- To collect data for problems related to the db2move command, go to the directory where you issued the command. Locate the following file(s), depending on the action you specified in the command:
  - For the COPY action, look for files called COPY.*timestamp*.ERR and COPYSCHEMA.*timestamp*.MSG. If you also specified either LOAD_ONLY or DDL_AND_LOAD mode, look for a file called LOADTABLE.*timestamp*.MSG as well.
  - For the EXPORT action, look for a file called EXPORT.out.
  - For the IMPORT action, look for a file called IMPORT.out.
  - For the LOAD action, look for a file called LOAD.out.
- To collect data for problems related to EXPORT or IMPORT or LOAD commands, determine whether your command included the MESSAGES parameter. If it did, collect the output file. These utilities use the current directory and the default drive as the destination if you do not specify otherwise.
- To collect data for problems related to a REDISTRIBUTE command, look for a file called *"databasename.database_partition_groupname. timestamp"* on Linux and UNIX and *"databasename. database_partition_groupname.date.time"* on Windows. It is located in *$HOME*/sqllib/db2dump directory or $DB2PATH\sqllib\redist respectively, where *$HOME* is the home directory of the instance owner.

© Copyright IBM Corp. 1993, 2008

Appendix E. Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command line help
  - Command help
  - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com®.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks publications online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. English DB2 Version 9.5 manuals in PDF format and translated versions can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg2700947.

Although the tables identify books available in print, the books might not be available in your country or region.

© Copyright IBM Corp. 1993, 2008

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The DB2 Information Center is updated more frequently than either the PDF or the hard-copy books.

Table 62. DB2 technical information

Name	Form Number	Available in print
Administrative API Reference	SC23-5842-01	Yes
Administrative Routines and Views	SC23-5843-01	No
Call Level Interface Guide and Reference, Volume 1	SC23-5844-01	Yes
Call Level Interface Guide and Reference, Volume 2	SC23-5845-01	Yes
Command Reference	SC23-5846-01	Yes
Data Movement Utilities Guide and Reference	SC23-5847-01	Yes
Data Recovery and High Availability Guide and Reference	SC23-5848-01	Yes
Data Servers, Databases, and Database Objects Guide	SC23-5849-01	Yes
Database Security Guide	SC23-5850-01	Yes
Developing ADO.NET and OLE DB Applications	SC23-5851-01	Yes
Developing Embedded SQL Applications	SC23-5852-01	Yes
Developing Java Applications	SC23-5853-01	Yes
Developing Perl and PHP Applications	SC23-5854-01	No
Developing User-defined Routines (SQL and External)	SC23-5855-01	Yes
Getting Started with Database Application Development	GC23-5856-01	Yes
Getting Started with DB2 installation and administration on Linux and Windows	GC23-5857-01	Yes
Internationalization Guide	SC23-5858-01	Yes
Message Reference, Volume 1	GI11-7855-00	No
Message Reference, Volume 2	GI11-7856-00	No
Migration Guide	GC23-5859-01	Yes
Net Search Extender Administration and User's Guide	SC23-8509-01	Yes
Partitioning and Clustering Guide	SC23-5860-01	Yes
Query Patroller Administration and User's Guide	SC23-8507-00	Yes
Quick Beginnings for IBM Data Server Clients	GC23-5863-01	No

Table 62. DB2 technical information (continued)

Name	Form Number	Available in print
Quick Beginnings for DB2 Servers	GC23-5864-01	Yes
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC23-8508-01	Yes
SQL Reference, Volume 1	SC23-5861-01	Yes
SQL Reference, Volume 2	SC23-5862-01	Yes
System Monitor Guide and Reference	SC23-5865-01	Yes
Troubleshooting Guide	GI11-7857-01	No
Tuning Database Performance	SC23-5867-01	Yes
Visual Explain Tutorial	SC23-5868-00	No
What's New	SC23-5869-01	Yes
Workload Manager Guide and Reference	SC23-5870-01	Yes
pureXML Guide	SC23-5871-01	Yes
XQuery Reference	SC23-5872-01	No

Table 63. DB2 Connect-specific technical information

Name	Form Number	Available in print
Quick Beginnings for DB2 Connect Personal Edition	GC23-5839-01	Yes
Quick Beginnings for DB2 Connect Servers	GC23-5840-01	Yes
DB2 Connect User's Guide	SC23-5841-01	Yes

Table 64. Information Integration technical information

Name	Form Number	Available in print
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	Yes
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	Yes
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-01	No
Information Integration: SQL Replication Guide and Reference	SC19-1030-01	Yes
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	Yes

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* DVD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation DVD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation DVD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2luw/v9r5.

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  1. Locate the contact information for your local representative from one of the following Web sites:
     - The IBM directory of world wide contacts at www.ibm.com/planetwide
     - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
  2. When you call, specify that you want to order a DB2 publication.
  3. Provide your representative with the titles and form numbers of the books that you want to order. For titles and form numbers, see "DB2 technical library in hardcopy or PDF format" on page 443.

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

To invoke SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

For DB2 Version 9.5 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

- To display topics in your preferred language in the Internet Explorer browser:
  1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     - To add a new language to the list, click the **Add...** button.

       Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
     - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.
- To display topics in your preferred language in a Firefox or Mozilla browser:
  1. Select the button in the **Languages** section of the **Tools** —> **Options** —> **Advanced** dialog. The Languages panel is displayed in the Preferences window.
  2. Ensure your preferred language is specified as the first entry in the list of languages.
     - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
     - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
  3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. Non-Administrative and Non-Root DB2 Information Centers always run in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you would like to install, you can use the Update feature to obtain and install them

   Note: If your environment requires installing the DB2 Information Center updates on a machine that is not connected to the internet, you have to mirror the update site to a local file system using a machine that is connected to the internet and has the DB2 Information Center installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.
   If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.
3. Stop the stand-alone Information Center, and restart the DB2 Information Center on your computer.

Note: On Windows Vista, the commands listed below must be run as an administrator. To launch a command prompt or graphical tool with full administrator privileges, right-click on the shortcut and then select **Run as administrator**.

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv95 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the <Program Files>\IBM\DB2 Information Center\Version 9.5 directory, where <Program Files> represents the location of the Program Files directory.
     c. Navigate from the installation directory to the doc\bin directory.
     d. Run the help_start.bat file:
        ```
        help_start.bat
        ```
   - On Linux:

a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the /opt/ibm/db2ic/V9.5 directory.
b. Navigate from the installation directory to the doc/bin directory.
c. Run the help_start script:

```
help_start
```

The systems default Web browser launches to display the stand-alone Information Center.

3. Click the **Update** button ( ). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
   - On Windows, navigate to the installation directory's doc\bin directory, and run the help_end.bat file:

     ```
     help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use `Ctrl-C` or any other method to terminate help_start.bat.
   - On Linux, navigate to the installation directory's doc/bin directory, and run the help_end script:

     ```
     help_end
     ```

     Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the help_start script.
7. Restart the DB2 Information Center.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv95 start
     ```

The updated DB2 Information Center displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click on the title.

"pureXML" in *pureXML Guide*
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

"Visual Explain" in *Visual Explain Tutorial*
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation
Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site
Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/support.html

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial 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.

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.

Appendix F. Notices

This information was developed for products and services offered in the U.S.A.

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
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This document may provide links or references to non-IBM Web sites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources that may be referenced, accessible from, or linked from this document. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or

© IBM Corporation 1993, 2008

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. Any software provided by third parties is subject to the terms and conditions of the license that accompanies that software.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain 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.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

The following terms are trademarks or registered trademarks of the International Business Machines Corporation in the United States, other countries, or both.

pureXML	1-2-3
Informix	TotalStorage
DB2	REXX
AT	AIX
WebSphere	OS/390
DB2 Connect	DB2 Universal Database
z/OS	Redbooks
System i	IBM
Lotus	System/370
Tivoli	DRDA
MVS	System Storage
OS/400	DS6000
ibm.com	Enterprise Storage Server
DS8000	

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

ADMIN_CMD procedure
  supported commands
    EXPORT 27
    IMPORT 86
    LOAD 230
ADMIN_COPY_SCHEMA procedure
  overview 3
anyorder file type modifier 199, 261
APIs
  db2Export 36
  db2Import 109
  db2Load 261
  sqluexpr 36
  sqluimpr 109
application record
  PC/IXF 386
ASC data type descriptions 382
ASC file
  format 381
  sample 384
ASC import file type 63
automatic dictionary creation (ADC)
  during data movement 156
auxiliary storage objects
  XML data specifier 428

B

binarynumerics file type modifier 199, 261
bind files
  used by export, import, load 435
books
  printed
    ordering 446
buffered inserts
  import utility 57
building indexes 153

C

character strings
  delimiter 379
chardel file type modifier
  export 18, 36
  import 63, 109
  load 199, 261
code page file type modifier 199, 261
code pages
  conversion
    files 411
    when importing or loading PC/IXF data 411
  Export API 36
  EXPORT command 18
  Import API 109
  IMPORT command 63
  import utility considerations 425
  load utility considerations 425
coldel file type modifier
  export
    db2Export API 36
    EXPORT command 18
  import
    db2Import API 109
    IMPORT command 63
  load
    db2Load API 261
    LOAD command 199
column descriptor record
  PC/IXF 386
columns
  incompatible 411
  invalid values 411
  LBAC protected
    export considerations 12
    importing to 55
    load considerations 134
    loading 129
    priveleges and authorites required to export 8
  specifying for import 109
commands
  db2inidb 356
  db2look 362
  db2move 329
  db2relocatedb 358
  EXPORT 18, 27
  IMPORT 63, 86
  LIST TABLESPACES 308
  LOAD 199, 230
  LOAD QUERY 302
  RESTORE DATABASE 338
compound file type modifier 63, 109
compressed tables
  loading data into 156
compression dictionaries
  KEEPDICTIONARY option 156
  RESETDICTIONARY option 156
constraint violations
  checking
    using the SET INTEGRITY statement 164
constraints
  checking
    after load operations 162
continuation record type
  PC/IXF 386
CURSOR file type
  data movement 141

D

data
  distribution 146
  exporting 8
  importing 48
  label-based access control (LBAC)
    exporting 8
    loading 128, 129
  moving across platforms 374

© Copyright IBM Corp. 1993, 2008

data *(continued)*
transfer
across platforms 374
between hosts and workstations 323
data movement
tools 3
Data Movement Guide
overview v
data record type
PC/IXF 386
data types
ASC 382
DEL 377
PC/IXF 402, 407
database movement tool command 329
databases
exporting from table into file
db2Export API 36
EXPORT command 18
importing from file into table
db2Import API 109
IMPORT command 63
loading data into tables 199
rebuilding
RESTORE DATABASE command 338
restoring 338
dateformat file type modifier
db2Import API 109
db2Load API 261
IMPORT command 63
LOAD command 199
DB2 Connect
moving data 323
DB2 Information Center
languages 447
updating 448
versions 447
viewing in different languages 447
DB2 statistics and DDL extraction tool command 362
db2inidb command
description 356
overview 355
db2Load API
description 261
DB2LOADREC registry variable
recovering data 176
db2look command
description 362
db2move command
description 329
overview 3
schema copying examples 328
db2relocatedb command
description 358
overview 3
DB2SECURITYLABEL data type
exporting 12
importing 55
loading 134
decplusblank file type modifier
EXPORT command 18
IMPORT command 63
LOAD command 199
decpt file type modifier
EXPORT command 18
IMPORT command 63
LOAD command 199
DEL data type descriptions 377
DEL file
format 375
sample 379
delimited ASCII (DEL) file format
moving data across platforms 374
overview 375
delimiters
character string 379
modifying 380
restrictions on moving data 380
delprioritychar file type modifier
IMPORT command 63
LBAC-protected data import 55
LBAC-protected data load 134
LOAD command 199
diagnostic information
data movement problems 441
distribution keys
loading data 180
documentation
overview 443
PDF 443
printed 443
terms and conditions of use 450
dump files
load utility 178
dumpfile file type modifier 199

E

exception tables
load utility 173
SET INTEGRITY statement 285
export API 36
EXPORT command
description
with ADMIN_CMD procedure 27
without ADMIN_CMD procedure 18
export utility
authorities required 8
file formats 373
identity columns 16
large objects (LOBs) 17
options 7
overview 3, 7
performance 7
prerequisites 8
privileges required 8
restrictions 8
table re-creating 13
transferring data between hosts and workstations 323
exported tables
re-creating 51
exporting
data
db2Export API 36
examples 42
EXPORT command 18
export utility overview 7
file type modifiers 18, 36
LBAC-protected 12
procedure 8
XML 9

F

fastparse file type modifier 199, 261
file formats
  CURSOR 141
  delimited ASCII (DEL) 375
  exporting table to file 18
  importing file to table 63
  nondelimited ASCII (ASC) 381
  PC version of IXF (PC/IXF) 385
  worksheet (WSF) 422
file type modifiers
  dumpfile 178
  Export API 36
  EXPORT utility 18
  Import API 109
  IMPORT command 63
  Load API 261
  LOAD command 199
forcein file type modifier 63, 109, 199, 261, 416

G

generated columns
  import utility 59
  using load utility 138
generatedignore file type modifier 59, 63, 109, 199, 261
generatedmissing file type modifier 59, 63, 109, 199, 261
generatedoverride file type modifier 199, 261

H

header record
  PC/IXF 386
help
  configuring language 447
  SQL statements 446
hierarchy records
  description 386

I

IBM Relational Data Replication Tools
  components 325
identity columns
  exporting data with 16
  import utility 58
  using load utility 136
identity record
  PC/IXF 386
identityignore 63
  file type modifier 109, 199, 261
identityignore file type modifier 58
identitymissing
  file type modifier 63, 109, 199, 261
identitymissing file type modifier 58
identityoverride
  file type modifier 199, 261
implieddecimal file type modifier 63, 109, 199, 261
Import API 109
IMPORT command 63
  using ADMIN_CMD 86
import operations
  ALLOW NO ACCESS 62
  ALLOW WRITE ACCESS 62
import utility
  authorities and privileges required to use 47
  buffered inserts 57
  client/server 61
  code page considerations 425
  compared to load utility 433
  file formats 373
  generated columns 59
  identity columns 58
  large objects (LOBs) 60
  overview 3, 45
  prerequisites 48
  re-creating an exported table 51
  remote database 61
  restrictions 48
  table locking 62
  transferring data between host and workstation 323
  user-defined distinct types (UDTs) 61
importing
  code page considerations 109
  data 48, 63
    LBAC protected 55
  database access through DB2 Connect 109
  effect of LBAC protection 47
  file to database table 109
  file type modifiers for 109
  overview 45
  PC/IXF files, data type-specific rules 412
  PC/IXF files, general rules 411
  PC/IXF files, with forcein 416
  PC/IXF, multiple-part files 109
  required information 45
  restrictions 109
  to a remote database 109
  to a table or hierarchy that does not exist 109
  to typed tables 109
  XML data 50
incompatible columns 411
index creation
  improving performance after load operations 153
indexes
  building 153
  modes 153
  PC/IXF record 386
  rebuilding 153
indexfreespace file type modifier 199, 261
indexixf file type modifier 63, 109
indexschema file type modifier 63, 109
Initialize a Mirrored Database command 356
Integration Exchange Format (IXF) 385
integrity checking 162

K

keepblanks file type modifier
  db2Import API 109
  IMPORT command 63
  loading
    db2Load API 261
    LOAD command 199

L

label-based access control (LBAC)
  exporting data 8

label-based access control (LBAC) *(continued)*
loading data
authorities and permissions 128
overview 129
protected data load considerations 134
protected importing
export 12
import 55
large objects (LOBs)
exporting 17
importing 60
LBAC (label-based access control)
exporting data 8, 12
importing data protected by 55
loading 129
loading data 128
loading data protected by 134
protected data
exporting 8
importing 47
loading 128, 129
LIST TABLESPACES command 308
Load API 261
LOAD command
in a partitioned database environment 181, 191
overview 199
using ADMIN_CMD 230
load copy location file 176
LOAD database authority 129
load delete start compensation log record 179
load operations
build phase 153
load pending list log record 179
LOAD QUERY command 302
in a partitioned database environment 188
load start log record
utility logs 179
load utility
authorities and privileges required to use 128
build phase 125
code page considerations 425
compared to import utility 433
data movement options 3
database recovery 125
delete phase 125
dump file 178
exception tables 173
features for maintaining referential integrity
overview 161
table space states 170
table states 172
file formats 373
file type modifiers 157
file type modifiers for 261
generated columns 138
identity columns 136
improving index creation 153
index copy phase 125
load phase 125
log records 179
moving data using SOURCEUSEREXIT 146
Not Load Restartable loads 174
optimizing performance 157
options for improving performance 157
overview 125
parallelism 152
prerequisites 129
load utility *(continued)*
recovery from failure 174
rejected rows 178
restarting failed loads 174
restrictions 129
table locking 167
table space states 170
table states 172
temporary files
LOAD command 199
overview 178
loading
access options 168
compressed tables 156
configuration options 195
data
LBAC protected 134
database partitions 186
examples
overview 282
partitioned database environments 192
partitioned database sessions 192
file to database table 199
file type modifiers for 199
into database partitions 180
into partitioned tables 132
multidimensional clustered (MDC) tables 145
overview 125
partitioned database environments 195
required information 125
table access options 168
using CURSOR 141
XML data 131
LOB Location Specifier (LLS) 385
LOBs (large objects)
exporting 17
importing 60
importing and exporting 427
lobsinfile file type modifier
Export API 36
export considerations 17
exporting 18
importing 63
loading 199
loading data into tables 261
loading overview 109
lobsinsepfiles file type modifier 17
locks
import utility 62
table level 167
log records
load utility 179

M

materialized query tables (MQTs)
dependent immediate 144
refreshing data 144
Set Integrity Pending state 144
message files
export, import, and load 7, 45, 125
modifiers
file type
EXPORT command 18
IMPORT command 63
LOAD command 199

moving data
between databases 63, 109
considerations for moving XML data 426
delimiter restrictions 380
export utility 7
import utility 45
load utility 125
using DB2 Connect 323
MQTs (materialized query tables)
dependent immediate 144
refreshing data 144
multidimensional clustering (MDC) tables
load considerations 145

N

nochecklengths file type modifier
importing 63
importing data to a table 109
loading 199
loading data in a table 261
nodefaults file type modifier
importing 63
importing data to a table 109
nodoubledel file type modifier
exporting 18
exporting to tables 109
importing 63
importing from tables 36
loading 199
loading tables 261
noeofchar file type modifier
importing 63
importing data into tables 109
loading 199
loading data into tables 261
noheader file type modifier
loading 199
loading data into tables 261
non-identity generated columns 59
nondelimited ASCII (ASC) file format 381
nonidentity generated columns 138
nonrecoverable databases
load options 125
norowwarnings file type modifier
LOAD command 199
loading data into tables 261
notices 453
notypeid file type modifier
IMPORT command 63
importing data into tables 109
nullindchar file type modifier
IMPORT command 199
importing data to tables 109
LOAD command 63
loading data to tables 261

O

options
forcein 416
ordering DB2 books 446

P

packeddecimal file type modifier 199
pagefreespace file type modifier 199
parallelism
load utility 152
partitioned database environments
loading data
migration 191
monitoring 188
overview 180, 186
restrictions 181
version compatibility 191
migrating 191
version compatibility 191
partitioned tables
loading 132
PC/IXF
code page conversion files 411
column values
invalid 411
data types
invalid 402, 411
valid 402, 407
file import
data type-specific rules 412
forcein option 416
general rules 411
moving data across platforms 374
overview 385
record types 386
System/370 IXF comparison 415
performance
load utility 157
privileges
export utility 8
import utility 47
load utility 128
problem determination
information available 450
tutorials 450

R

rebuilding
indexes 153
reclen file type modifier 63
importing 109
Load API 261
loading 199
record types
PC/IXF 386
recoverable databases
load options 125
recovery
database 338
without roll forward 338
redirected restore
using generated script 337
registry variables
DB2LOADREC 176
Relocate Database command 358
REMOTEFETCH media type
data movement 141
replication tools 325
restarting a load operation
allow read access mode 176

restarting a load operation *(continued)*
multi-partition database load operations 189
restarting load operations 174
RESTORE DATABASE command 338
restore utility
GENERATE SCRIPT option
overview 3
REDIRECT option
overview 3
restoring
earlier versions of DB2 databases 338
rollforward utility
load copy location file 176
rows
exporting LBAC protected 8, 12
importing to LBAC protected 55
loading data into LBAC-protected 134
loading into LBAC protected 129

S

samples
files
ASC 384
DEL 379
schemas
copying 326
troubleshooting tips 326
seclabelchar file type modifier 55, 134
seclabelname file type modifier 55, 134
SELECT statement
in EXPORT command 18
semantics
forcein
code page 416
data type 416
general 416
SET CONSTRAINTS statement 285
set integrity pending state 285
SET INTEGRITY statement 285
checking for constraint violations 164
SOURCEUSEREXIT option
data movement 146
split mirror
handling 355
overview 3
SQL statements
displaying help 446
SET CONSTRAINTS 285
SET INTEGRITY 285
sqluexpr API 36
sqluimpr API 109
staging tables
dependent immediate 143
propagating 143
storage
XML data specifier 428
striptblanks file type modifier 55, 63, 109, 134, 199, 261
striptnulls file type modifier 63, 109, 199, 261
structure
delimited ASCII (DEL) files 375
non-delimited ASCII (ASC) files 381
subtable record
PC/IXF 386
subtableconvert file type modifier 199
summary tables
import restriction 48
suspended I/O to support continuous availability 355
syntax
description 437
System/370 IXF
contrasted with PC/IXF 415
contrasted with System/370 415

T

table load delete start log record 179
table record
PC/IXF 386
table space states
Backup Pending 170
Load in Progress 170
Normal 170
Restore Pending 170
table spaces
states 170
table states
Load in Progress 172
Load Pending 172
Normal 172
Not Load Restartable 172
Read Access Only 172
Set Integrity Pending 172
Unavailable 172
tables
exception 285
exported, re-creating 51
exporting to files 18, 36
importing files 63, 109
loading files to 199
locking 167
temporary files
LOAD command 199
load utility 178
termination
load operations 176
in multi-partition databases 189
records
PC/IXF 386
terms and conditions
use of publications 450
timeformat file type modifier 63, 109, 199, 261
timestampformat file type modifier
db2import API 109
db2load API 261
IMPORT command 63
LOAD command 199
totalfreespace file type modifier 199, 261
troubleshooting
diagnostic data
for data movement 441
online information 450
tutorials 450
tutorials
problem determination 450
troubleshooting 450
Visual Explain 449
typed tables
exporting 14
importing 52
moving data between 14, 52
re-creating 52
traverse order 14, 52

U

Unicode (UCS-2)
  data movement considerations 423
updates
  DB2 Information Center 448
usedefaults file type modifier 55, 63, 109, 134, 199, 261
user exit programs
  customize 146
  data movement 146
user-defined types
  distinct types
    importing 61
utilities
  file formats 373

V

Visual Explain
  tutorial 449

W

Worksheet File Format (WSF)
  description 422
  See WSF (Worksheet File Format) 422
WSF (Worksheet File Format)
  description 422
  moving data across platforms 374

X

XML
  data type
    importing and exporting 427
XML data
  considerations for moving 426
  exporting 9
  importing 50
  loading 131
  movement 425
  Query and XPath Data Model 429
XQuery statements
  Query and XPath Data Model 429

Z

zoned decimal file type modifier 199, 261

Printed in USA

SC23-5847-01

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

Data Movement Utilities Guide and Reference