IBM® DB2 Universal Database™

Data Movement Utilities Guide and Reference

Version 8

SC09-4830-00

IBM® DB2 Universal Database™

Data Movement Utilities Guide and Reference

Version 8

SC09-4830-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

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).

Contents

About This Book

This book provides information about, and shows you how to use, the following IBM DB2 Universal Database (UDB) data movement utilities:

- The Import and Export 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.
- The Load utility moves data into tables, extends existing indexes, and generates statistics. Load moves data much faster than the import utility when large amounts of data are involved. Data unloaded using the export utility can be loaded with the load utility.
- When the Load utility is used in a partitioned database environment, large amounts of data can be partitioned and loaded into different database partitions.
- DataPropagator (DPROP) is a component of DB2 Universal Database that allows automatic copying of table updates to other tables in other DB2 relational databases.
- The Data Warehouse Center (DWC) can be used to move data from operational databases to a warehouse database.

Other vendor's products that move data in and out of databases are also available, but are not discussed in this book.

Who Should Use this Book

This manual is for database administrators, application programmers, and other DB2 UDB users who perform the following tasks:

- Load data into DB2 tables from operating system files
- Move data between DB2 databases, and between DB2 and other applications (for example, spreadsheets)
- Archive data.

It is assumed that you are familiar with DB2 Universal Database, Structured Query Language (SQL), and with the operating system environment in which DB2 UDB is running.

How this Book is Structured

The following topics are covered:

Chapter 1
Describes the DB2 export utility, used to move data from DB2 tables into files.

Chapter 2
Describes the DB2 import utility, used to move data from files into DB2 tables or views.

Chapter 3
Describes the DB2 load utility, used to move large volumes of data into DB2 tables.

Chapter 4
Describes loading data in a partitioned database environment.

Chapter 5
Describes how to use the DB2 export, import, and load utilities to move DB2 Data Links Manager data.

Chapter 6
Describes how to use the DB2 export, import, and load utilities to transfer data across platforms, and to and from DRDA host databases. DataPropagator (DPROP), another method for moving data between databases in an enterprise, is also described. Also introduces the Data Warehouse Center (DWC), which can be used to move data from operational databases to a warehouse database.

Appendix A
Explains the conventions used in syntax diagrams.

Appendix B
Summarizes the important differences between the DB2 load and import utilities.

Appendix C
Includes an API sample program that illustrates how to export data to a file, import data to a table, load data into a table, and check the status of a load operation.

Appendix D
Describes external file formats supported by the database manager export, import, and load utilities.

Appendix E
Discusses unicode consideration when using the export, import and load utilities.

Appendix F
Lists bind files with their default isolation levels, as well as which utilities use them and for what purpose.

Appendix G

Provides information about interpreting messages generated by the database manager when a warning or error condition has been detected.

Chapter 1. Export

This chapter describes the DB2 UDB export utility, which is used to write data from a DB2 database to one or more files stored outside of the database. The exported data can then be imported or loaded into another DB2 database, using the DB2 import or the DB2 load utility, respectively, or it can be imported into another application (for example, a spreadsheet).

The following topics are covered:

- "Export Overview"
- "Privileges, Authorities and Authorization Required to Use Export" on page 3
- "Using Export" on page 3
- "Using export with identity columns" on page 4
- "Recreating an exported table" on page 5
- "Exporting large objects (LOBS)" on page 5
- "Exporting data in parallel" on page 6
- "EXPORT" on page 8
- "sqluexpr - Export" on page 17
- "SQLUEXPT-OUT" on page 29
- "Export Sessions - CLP Examples" on page 30

For information about exporting DB2 Data Links Manager data, see "Using Export to Move DB2 Data Links Manager Data" on page 279. For information about exporting data out of typed tables, see "Moving Data Between Typed Tables" on page 293. For information about exporting data from a DRDA server database to a file on the DB2 Connect workstation, and the reverse, see "Moving Data With DB2 Connect" on page 285.

Export Overview

The export utility exports data from a database to an operating system file, which can be in one of several external file formats. This operating system file can then be used to move the table data to a different server such as DB2® UDB for iSeries.

The following information is required when exporting data:

- An SQL SELECT statement specifying the data to be exported.

- The path and name of the operating system file that will store the exported data.
- The format of the data in the input file. This format can be IXF, WSF, or DEL.
- When exporting typed tables, you may need to provide the subtable traverse order within the hierarchy. If the IXF format is to be used, the default order is recommended. When specifying the order, recall that the subtables must be traversed in the PRE-ORDER fashion. When exporting typed tables, you cannot provide a SELECT statement directly. Instead, you must specify the target subtable name, and optionally a WHERE clause. The export utility uses this information, along with the traverse order, to generate and execute the required SELECT statement.

You can also specify:

- New column names when exporting to IXF or WSF files. If you do not want to specify new column names, the column names in the existing table or view are used in the exported file.
- Additional options to customize the export operation.
- A message file name. During DB2 operations such as exporting, importing, loading, binding, or restoring data, 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. These message files are standard ASCII text files. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

If you want to use the export utility in a multiple database partition environment, you can use **db2batch** to complete the task at each database partition. The SELECT statement must be able to return only the data found locally. The selection condition is as follows:

```
SELECT * FROM tablename WHERE NODENUMBER(column-name) = CURRENT NODE
```

Related concepts:

- “Privileges, Authorities and Authorization Required to Use Export” on page 3
- “Using export with identity columns” on page 4
- “Recreating an exported table” on page 5
- “Exporting large objects (LOBS)” on page 5
- “Exporting data in parallel” on page 6
- “Moving Data Between Typed Tables” on page 293
- “Examples of db2batch tests” in the *Administration Guide: Performance*

Related tasks:

- "Using Export" on page 3

Related reference:

- "sqluexpr - Export" on page 17
- "db2batch - Benchmark Tool" in the *Command Reference*
- "Export Sessions - CLP Examples" on page 30
- "Export/Import/Load Utility File Formats" on page 323
- "EXPORT" on page 8

Privileges, Authorities and Authorization Required to Use Export

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.

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

Related reference:

- "sqluexpr - Export" on page 17
- "EXPORT" on page 8

Using Export

Prerequisites:

Before invoking the export utility, you must be connected to (or be able to implicitly connect to) the database from which the data will be exported. Since the utility will issue a COMMIT statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking export. Other user applications accessing the table using separate connections need not disconnect.

Restrictions:

The following restrictions apply to the export utility:

- This utility does not support the use of nicknames.
- This utility does not support tables with structured type columns.

Procedure:

The export utility can be invoked through the command line processor (CLP), the Export notebook in the Control Centre, or an application programming interface (API), **sqluexpr***.*

Following is an example of the EXPORT command issued through the CLP:

```
db2 export to staff.ixf of ixf select * from userid.staff
```

To open the Export notebook:

1. From the Control Center, expand the object tree until you find the Tables or Views folder.
2. Click on the folder 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. Click the right mouse button on the table or view you want in the contents pane, and select Export from the pop-up menu. The Export notebook opens.

Detailed information about the Control Center is provided through its online help facility.

Related samples:

- "tbmove.out -- HOW TO MOVE TABLE DATA (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.out -- HOW TO MOVE TABLE DATA (C++)"
- "tbmove.sqC -- How to move table data (C++)"

Using export with identity columns

The export utility can be used to export data from a table containing an identity column. If the SELECT statement specified for the export operation is of the form "select * from tablename", and the METHOD option is not used, exporting identity column properties to IXF files is supported. The REPLACE_CREATE and the CREATE options of the IMPORT command can then be used to recreate 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).

Recreating an exported table

A table can be saved by using the export utility and specifying the IXF file format. The saved table (including its indexes) can then be recreated using the import utility.

The export operation will fail if the data you want to export exceeds the space available on the file system on which the exported file will be created. In this case, you should limit the amount of data selected by specifying conditions on the WHERE clause, so that the export file will fit on the target file system. You can invoke the export utility multiple times to export all of the data.

The DEL and ASC file formats do not contain descriptions of the target table, but they do contain the record data. To recreate a table with data stored in these file formats, create the target table, and then use the load, or import utility to populate the table from these files. **db2look** (DB2 Statistics and DDL Extraction Tool) can be used to capture the original table definitions, and to generate the corresponding data definition language (DDL).

Related concepts:

- "Using Import to Recreate an Exported Table" on page 39

Related reference:

- "db2look - DB2 Statistics and DDL Extraction Tool" in the *Command Reference*

Exporting large objects (LOBS)

When exporting data from large object (LOB) columns, the default action is to select the first 32KB of data, and to place this data in the same file as the rest of the column data.

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 recreate a table containing a LOB column that is defined to be 1GB or larger.

A LOB Location Specifier (LLS) is used to store multiple LOBs in a single file when exporting LOB information. When exporting data using the `lobsinfile` modifier, the export utility selects the entire LOB file and places it in one of the LOB files. There may be multiple LOBs per LOB file and multiple LOB files in each LOB path. The data file will contain the LLS records.

An LLS is a string indicating where LOB data can be found within a file. The format of the LLS is `filename.ext:nnn:mmm/`, where `filename.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 (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.

Related reference:

- "EXPORT" on page 8

Exporting data in parallel

Exporting data in parallel reduces data transfer, and distributes the writing of the result set, as well as the generation of the formatted output, across nodes in a more effective manner than would otherwise be the case. When data is exported in parallel (by invoking multiple export operations, one for each partition of a table), it is extracted, converted on the local nodes, and then written to the local file system. In contrast, when exporting data serially (exporting through a single export operation), it is extracted in parallel and then shipped to the client, where a single process performs conversion and writes the result set to a local file system.

The **db2batch** command is used to monitor the performance characteristics and execution duration of SQL statements. This utility also has a parallel export function in partitioned database environments that:

- Runs queries to define the data to be exported
- On each partition, creates a file containing the exported data that resides on that partition.

There are two types of parallel export, depending on what tables are queried, and where those tables exist in the partitioned database environment:

- Parallel export from a single partition table, or a join or subquery on multiple tables that are collocated (specify `-p s` on the **db2batch** command).

  There are two ways in which tables can be considered collocated:

  - The tables are in a single database partition group defined on the same partition.
  - The tables are in the same database partition group, and have partitioning keys with the same number and type of columns. The

corresponding columns of the partitioning key are partition compatible, and the tables are equi-joined on the entire partition key, or a superset of the partition key.

In each case, the query can be run on each partition to generate that partition's export data file using the NODENUMBER function as described below. (Note that if a table exists in a single partition only, export parallelism is negated, because the data is retrieved from only one partition. To enable export parallelism in this case, refer to the next bullet.)

- A SELECT from multiple non-collocated tables (specify `-p t` *`tablename`* or `-p d` on the **db2batch** command; the former allows you to specify the name of an existing table to use as a staging table, while the latter causes the export utility to create a staging table).

  The export utility uses a staging table that is populated through the export query. This staging table is used to locate the rows of the ″export″ result set by running an INSERT of the fullselect query into the staging table. Once the staging table is created, the export utility generates an export data file at each partition by running:

  ```
  "select * WHERE NODENUMBER(colname) = CURRENT NODE"
  ```

  on the staging table.

  A staging table can also be used to export a single partition table in parallel. In most cases, transferring the data from a single partition into a multi-partition staging table, and then exporting the staging table in parallel on all partitions, is faster than exporting the single partition table serially.

The export utility runs a query in parallel on each partition to retrieve the data on that partition. In the case of `db2batch -p s`, the original SELECT query is run in parallel. In the case of `db2batch -p t` and `db2batch -p d`, a staging table is loaded with the export data, using the specified query, and a `SELECT *` query is run on the staging table in parallel on each partition to export the data. To export only the data that resides on a given partition, **db2batch** adds the predicate `NODENUMBER(`*`colname`*`) = CURRENT NODE` to the WHERE clause of the query that is run on that partition. The *colname* parameter must be set to the qualified or the unqualified name of a table column. The export utility uses the first column name in the original query to set this parameter.

Following are the limitations on queries used by the export utility:

- When specifying `db2batch -p s`, the query must not contain only column functions, because a column name is needed for the `NODENUMBER` *`colname`* predicate.
- When specifying `db2batch -p s`, aggregates (such as min, max, and avg) must be based on a grouping that includes the partitioning key.

- When specifying `db2batch -p t` or `db2batch -p d`, the query cannot contain ORDER BY, because ORDER BY on a fullselect within an INSERT statement is not supported by DB2® UDB.

If, when specifying `-p s` on the **db2batch** command, you also use the `-r` option to create result output files, the files on each partition will be in sorted order if you have an ORDER BY clause. If a single sorted file is your objective, merge the sorted file on each partition into one sorted file. For example, on UNIX® based systems, use the command `sort -m` to merge the files into a single sorted file. If you are sending your output to an NFS mounted file system, the output will not be sorted, even if you specify the ORDER BY clause.

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.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

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

Required connection:

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

Command syntax:

```
►►──EXPORT TO──filename──OF──filetype──┬─────────────────────────┬──────────►
                                       │          ┌─,────────┐   │
                                       │          ▼          │   │
                                       └─LOBS TO────lob-path─┴───┘

►──┬──────────────────────────┬──┬───────────────────────────────┬──────────►
   │          ┌─,────────┐    │  │              ┌─────────────┐  │
   │          ▼          │    │  │              ▼             │  │
   └─LOBFILE────filename─┴────┘  └─MODIFIED BY────filetype-mod─┴──┘
```

```
►──┬──────────────────────────────────────┬──┬──────────────────────────────┬──────────►
   │                ┌─,─────────────┐     │  └─MESSAGES──message-file─┘
   │                ▼               │     │
   └─METHOD N──(────column-name─────┴──)──┘

►──┬─select-statement──────────────────────────────────────────────────┬──────────────►◄
   └─HIERARCHY───┬─STARTING──sub-table-name───┬──┬─────────────────┬───┘
                 └─┤ traversal-order-list ├───┘  └─where-clause────┘
```

traversal-order-list:

```
      ┌─,─────────────────┐
      ▼                   │
├──(──sub-table-name──────┴──)───────────────────────────────────────────────────┤
```

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.

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. 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`, `/u/foo/lob/path/bar.002`, 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.

MESSAGES message-file
Specifies the destination for warning and error messages that occur

during an export operation. If the file already exists, the export utility appends the information. If *message-file* is omitted, the messages are written to standard output.

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 additional options (see Table 1 on page 13).

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

  Note: 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 may result in incorrect data, depending on the operating system.
- IXF (integrated exchange format, PC version), in which most of the table attributes, as well as any existing indexes, are saved in the IXF file, except when columns are specified in the SELECT statement. With this format, the table can be recreated, while with the other file formats, the table must already exist before data can be imported into it.

select-statement
Specifies the SELECT statement that will return the data to be exported. If the SELECT 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
Specifies the name of the file to which data is to be exported. If the complete path to the file is not specified, the export utility uses the current directory and the default drive as the destination.

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.

Examples:

The following example shows how to export information from the STAFF table in the SAMPLE database to the file `myfile.ixf`. The output will be in IXF format. Note that you must be connected to the SAMPLE database before issuing the command. The index definitions (if any) will be stored in the output file except when the database connection is made through DB2 Connect.

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

The following example shows how to export the information about employees in Department 20 from the STAFF table in the SAMPLE database. The output will be in IXF format and will go into the awards.ixf file. Note that you must first connect to the SAMPLE database before issuing the command. Also note that the actual column name in the table is ′dept′ instead of ′department′.

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

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
```

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

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

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
```

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.

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

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.

DB2 Data Links Manager Considerations

To ensure that a consistent copy of the table and the corresponding files referenced by the DATALINK columns are copied for export, do the following:

1. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename SHARE.

This ensures that no update transactions are in progress when EXPORT is run.

2. Issue the EXPORT command.
3. Run the **dlfm_export** utility at each Data Links server. Input to the **dlfm_export** utility is the control file name, which is generated by the export utility. This produces a tar (or equivalent) archive of the files listed within the control file. For Distributed File Systems (DFS), the **dlfm_export** utility will get the DCE network root credentials before archiving the files listed in the control file. **dlfm_export** does not capture the ACLs information of the files that are archived.
4. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename RESET.

   This makes the table available for updates.

EXPORT is executed as an SQL application. The rows and columns satisfying the SELECT statement conditions are extracted from the database. For the DATALINK columns, the SELECT statement should not specify any scalar function.

Successful execution of EXPORT results in generation of the following files:

- An export data file as specified in the EXPORT command. A DATALINK column value in this file has the same format as that used by the IMPORT and LOAD utilities. When the DATALINK column value is the SQL NULL value, handling is the same as that for other data types.
- Control files *server_name*, which are generated for each Data Links server. On Windows operating systems, a single control file, `ctrlfile.lst`, is used by all Data Links servers. For DFS, there is one control file for each cell. These control files are placed in the directory <data-file path>/dlfm/YYYYMMDD/HHMMSS (on the Windows NT operating system, `ctrlfile.lst` is placed in the directory <data-file path>\dlfm\YYYYMMDD\HHMMSS). YYYYMMDD represents the date (year month day), and HHMMSS represents the time (hour minute second).

The **dlfm_export** utility is provided to export files from a Data Links server. This utility generates an archive file, which can be used to restore files in the target Data Links server.

Table 1. Valid File Type Modifiers (Export)

Modifier	**Description**
All File Formats	

Table 1. Valid File Type Modifiers (Export) (continued)

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. 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/.
DEL (Delimited ASCII) File Format	

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.[a] 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. **Note:** The CODEPAGE modifier cannot be used with the LOBSINFILE modifier.

Table 1. Valid File Type Modifiers (Export) (continued)

Modifier	Description
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.[a] In the following example, `coldel;` causes the export utility to interpret any semicolon (;) it encounters as a column delimiter: `db2 "export to temp of del modified by coldel; select * from staff where dept = 20"`
datesiso	Date format. Causes all date data values to be exported in ISO format ("*YYYY-MM-DD*").[b]
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.[a]
dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [a] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.
nodoubledel	Suppresses recognition of double character delimiters. For more information, see "Delimiter Restrictions" on page 16.
WSF File Format	
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[c] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[c]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[c]
4	Creates a WSF file containing DBCS characters.

Table 1. Valid File Type Modifiers (Export) (continued)

Modifier	Description

Notes:

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. [a] “Delimiter Restrictions” lists restrictions that apply to the characters that can be used as delimiter overrides.
3. [b] 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. [c] 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 may be specified.

Delimiter Restrictions

It is the user’s responsibility to ensure that the chosen delimiter character is not part of the data to be moved. If it is, unexpected errors may occur. The following restrictions apply to column, string, DATALINK, and decimal point delimiters when moving data:

- Delimiters are mutually exclusive.
- A delimiter cannot be binary zero, a line-feed character, a carriage-return, or a blank space.
- The default decimal point (.) cannot be a string delimiter.
- 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:

    ```
    " (0x22, double quotation mark; string delimiter)
    , (0x2c, comma; column delimiter)
    ```
  - Default delimiters for data files in EBCDIC SBCS code pages are:

```
" (0x7F, double quotation mark; string delimiter)
, (0x6B, comma; column delimiter)
```

- 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 ...
  ```

The following information about support for double character delimiter recognition in DEL files applies to the export, import, and load utilities:

- Character delimiters are permitted within the character-based fields of a DEL file. This applies to fields of type CHAR, VARCHAR, LONG VARCHAR, or CLOB (except when `lobsinfile` is specified). Any pair of character delimiters found between the enclosing character delimiters is imported or loaded into the database. For example,

  ```
  "What a ""nice"" day!"
  ```

  will be imported as:

  ```
  What a "nice" day!
  ```

  In the case of export, the rule applies in reverse. For example,

  ```
  I am 6" tall.
  ```

  will be exported to a DEL file as:

  ```
  "I am 6"" tall."
  ```

- In a DBCS environment, the pipe (|) character delimiter is not supported.

Related concepts:

- "Export Overview" on page 1

sqluexpr - 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.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

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

Required connection:

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

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluexpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqluexpr (
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    sqlu_media_list *pLobFileList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluexpt_out *pOutputInfo,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgexpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgexpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    sqlu_media_list *pLobFileList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluexpt_out *pOutputInfo,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameter:

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

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

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

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

pLobPathList
Input. An *sqlu_media_list* using *media_type* SQLU_LOCAL_MEDIA, and the *sqlu_media_entry* structure listing paths on the client where the LOB files are to be stored.

When file space is exhausted on the first path in this list, the API will use the second path, and so on.

pLobFileList
Input. An *sqlu_media_list* using *media_type* SQLU_CLIENT_LOCATION, and the *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 *pLobFilePath*), and then appending a 3-digit sequence number. For example, if the current LOB path is the directory `/u/foo/lob/path`, and the current LOB file name is `bar`, the created LOB files will be `/u/foo/lob/path/bar.001`, `/u/foo/lob/pah/bar.002`, and so on.

pDataDescriptor

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`) 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 *pActionString*.

pActionString

Input. Pointer to an *sqlchar* structure containing a valid dynamic SQL SELECT statement. The structure contains a 2-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 *pDataDescriptor*), 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.

pFileType

Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in `sqlutil`) 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 Integrated 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.

pFileTypeMod
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.

pMsgFileName
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 overwritten. If it does not exist, a file is created.

CallerAction
Input. An action requested by the caller. Valid values (defined in `sqlutil`) 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.

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

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

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 Integrated 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.

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.

A warning message is issued if the number of columns (*dcolnum*) in the external column name array, *pDataDescriptor*, 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.

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.

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.

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 *pActionString* beginning with `SELECT * FROM tablename`, and the *pDataDescriptor* parameter specifying default names. Indexes are not saved for views, or if the SELECT clause of the *pActionString* includes a join. A WHERE clause, a GROUP BY clause, or a HAVING clause in the *pActionString* 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.

DB2 Data Links Manager Considerations

To ensure that a consistent copy of the table and the corresponding files referenced by the DATALINK columns are copied for export, do the following:

1. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename SHARE.

   This ensures that no update transactions are in progress when EXPORT is run.
2. Issue the EXPORT command.
3. Run the **dlfm_export** utility at each Data Links server. Input to the **dlfm_export** utility is the control file name, which is generated by the export utility. This produces a tar (or equivalent) archive of the files listed within the control file. For Distributed File Systems (DFS), the **dlfm_export** utility will get the DCE network root credentials before archiving the files

listed in the control file. **dlfm_export** does not capture the ACLs information of the files that are archived.

4. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename RESET.

   This makes the table available for updates.

EXPORT is executed as an SQL application. The rows and columns satisfying the SELECT statement conditions are extracted from the database. For the DATALINK columns, the SELECT statement should not specify any scalar function.

Successful execution of EXPORT results in generation of the following files:

- An export data file as specified in the EXPORT command. A DATALINK column value in this file has the same format as that used by the IMPORT and LOAD utilities. When the DATALINK column value is the SQL NULL value, handling is the same as that for other data types.
- Control files *server_name*, which are generated for each Data Links server. On the Windows NT operating system, a single control file, `ctrlfile.lst`, is used by all Data Links servers. For DFS, there is one control file for each cell. These control files are placed in the directory <data-file path>/dlfm/YYYYMMDD/HHMMSS (on the Windows NT operating system, `ctrlfile.lst` is placed in the directory <data-file path>\dlfm\YYYYMMDD\HHMMSS). YYYYMMDD represents the date (year month day), and HHMMSS represents the time (hour minute second).

The **dlfm_export** utility is provided to export files from a Data Links server. This utility generates an archive file, which can be used to restore files in the target Data Links server.

Table 2. Valid File Type Modifiers (Export)

Modifier	Description
All File Formats	
lobsinfile	*lob-path* specifies the path to the files containing LOB values.
DEL (Delimited ASCII) File Format	
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.[a] The single quotation mark (') can also be specified as a character string delimiter as follows: `modified by chardel''`

Table 2. Valid File Type Modifiers (Export) (continued)

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 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. **Note:** 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.[a]
	In the following example, `coldel;` causes the export utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 "export to temp of del modified by coldel;`
	`select * from staff where dept = 20"`

datesiso	Date format. Causes all date data values to be exported in ISO format ("*YYYY-MM-DD*").[b]
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.[a]
dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [a] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.
nodoubledel	Suppresses recognition of double character delimiters.[a]
WSF File Format	
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[c] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[c]

Table 2. Valid File Type Modifiers (Export) (continued)

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

Notes:

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. [a] 27 lists restrictions that apply to the characters that can be used as delimiter overrides.
3. [b] 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. [c] 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 may be specified.

Delimiter restrictions:

It is the user's responsibility to ensure that the chosen delimiter character is not part of the data to be moved. If it is, unexpected errors may occur. The following restrictions apply to column, string, DATALINK, and decimal point delimiters when moving data:

- Delimiters are mutually exclusive.
- A delimiter cannot be binary zero, a line-feed character, a carriage-return, or a blank space.
- The default decimal point (.) cannot be a string delimiter.
- 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:

  ```
  " (0x22, double quotation mark; string delimiter)
  , (0x2c, comma; column delimiter)
  ```
- Default delimiters for data files in EBCDIC SBCS code pages are:

  ```
  " (0x7F, double quotation mark; string delimiter)
  , (0x6B, comma; column delimiter)
  ```
- 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 ...
  ```

The following information about support for double character delimiter recognition in DEL files applies to the export, import, and load utilities:

- Character delimiters are permitted within the character-based fields of a DEL file. This applies to fields of type CHAR, VARCHAR, LONG VARCHAR, or CLOB (except when `lobsinfile` is specified). Any pair of character delimiters found between the enclosing character delimiters is imported or loaded into the database. For example,

  ```
  "What a ""nice"" day!"
  ```

  will be imported as:

  ```
  What a "nice" day!
  ```

  In the case of export, the rule applies in reverse. For example,

  ```
  I am 6" tall.
  ```

  will be exported to a DEL file as:

  ```
  "I am 6"" tall."
  ```
- In a DBCS environment, the pipe (|) character delimiter is not supported.

Related concepts:

- “Moving DB2 Data Links Manager Using Export - Concepts” on page 275

Related reference:

- “SQLCA” in the *Administrative API Reference*
- “SQLCHAR” in the *Administrative API Reference*
- “SQLDCOL” in the *Administrative API Reference*
- “SQLUEXPT-OUT” on page 29
- “SQLU-MEDIA-LIST” in the *Administrative API Reference*

Related samples:

- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tload.sqb -- How to export and load table data (IBM COBOL)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

SQLUEXPT-OUT

This structure is used to pass information from sqluexpr.

Table 3. Fields in the SQLUEXPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of the structure.
ROWSEXPORTED	INTEGER	Number of records exported from the database into the target file.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQL-UEXPT-OUT */
/* ... */
SQL_STRUCTURE sqluexpt_out
{
  sqluint32       sizeOfStruct;
  sqluint32       rowsExported;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UEXPT-OUT.
    05 SQL-SIZE-OF-UEXPT-OUT  PIC 9(9) COMP-5 VALUE 8.
    05 SQL-ROWSEXPORTED       PIC 9(9) COMP-5 VALUE 0.
*
```

Related reference:

- "sqluexpr - Export" on page 17

Export Sessions - CLP Examples

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
```

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
```

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
```

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

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

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
```

Related reference:

- "EXPORT" on page 8

Chapter 2. Import

This chapter describes the DB2 UDB import utility, which uses the SQL INSERT statement to write data from an input file into a table or view. If the target table or view already contains data, you can either replace or append to the existing data.

The following topics are covered:

- "Import Overview" on page 32
- "Privileges, Authorities, and Authorization Required to Use Import" on page 33
- "Using Import" on page 33
- "Using Import in a Client/Server Environment" on page 35
- "Using Import with Buffered Inserts" on page 36
- "Using Import with Identity Columns" on page 36
- "Using Import with Generated Columns" on page 38
- "Using Import to Recreate an Exported Table" on page 39
- "Importing Large Objects (LOBS)" on page 41
- "Importing User-defined Distinct Types (UDTs)" on page 42
- "Table Locking During Import" on page 42
- "IMPORT" on page 42
- "sqluimpr - Import" on page 67
- "SQLUIMPT-IN" on page 92
- "SQLUIMPT-OUT" on page 93
- "Character Set and NLS Considerations" on page 95
- "Import Sessions - CLP Examples" on page 95

For information about importing DB2 Data Links Manager data, see "Using Import to Move DB2 Data Links Manager Data" on page 279. For information about importing data from typed tables, see "Moving Data Between Typed Tables" on page 293. For information about importing data from a file on the DB2 Connect workstation to a DRDA server database, and the reverse, see "Moving Data With DB2 Connect" on page 285.

Import Overview

The import utility inserts data from an input file into a table or updatable view. If the table or view receiving the imported data already contains data, you can either replace or append to the existing data.

The following information is required when importing data:

- 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. This format can be IXF, WSF, DEL, or ASC.
- Whether the input data is to be inserted into the table or view, or whether existing data in the table or view is to be updated or replaced by the input data.
- A message file name, if the utility is invoked through the application programming interface (API), **sqluimpr**.
- When working with typed tables, you may need to provide the method or order by which to progress through all of the structured types. The order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy is called the *traverse* order. This order is important when moving data between table hierarchies, because it determines where the data is moved in relation to other data.

  When working with typed tables, you may also need to provide the subtable list. This list shows into which subtables and attributes to import data.

You can also specify:

- The method to use for importing the data: column location, column name, or relative column position.
- The number of rows to INSERT before committing the changes to the table. Requesting periodic COMMITs reduces the number of rows that are lost if a failure and a ROLLBACK occur during the import operation. It also prevents the DB2® logs from getting full when processing a large input file.
- The number of file records to skip before beginning the import operation. If an error occurs, you can restart the import operation immediately following the last row that was successfully imported and committed.
- The names of the columns within the table or view into which the data is to be inserted.
- A message file name. During DB2 operations such as exporting, importing, loading, binding, or restoring data, 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. These message files are standard ASCII text files.

Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

Note: Specifying target table column names or a specific importing method makes importing to a remote database slower.

Related concepts:

- "Moving Data Between Typed Tables" on page 293

Related reference:

- "sqluimpr - Import" on page 67
- "Import Sessions - CLP Examples" on page 95
- "Export/Import/Load Utility File Formats" on page 323
- "IMPORT" on page 42

Privileges, Authorities, and Authorization 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.

To use the import utility to create a new table, you must have SYSADM authority, DBADM authority, or CREATETAB privilege for the database. To replace data in an existing table or view, you must have SYSADM authority, DBADM authority, or CONTROL privilege for the table or view. To append data to an existing table or view, you must have SELECT and INSERT privileges for the table or view.

Related reference:

- "sqluimpr - Import" on page 67
- "IMPORT" on page 42

Using Import

Prerequisites:

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. Since the utility will issue a COMMIT or a ROLLBACK statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking import.

Restrictions:

The following restrictions apply to the import utility:

- This utility does not support the use of nicknames.
- 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 64KB may, in some cases, be exceeded.

The following limitation applies to the import utility:

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

Procedure:

The import utility can be invoked through the command line processor (CLP), the Import notebook in the Control Centre, or an application programming interface (API), ***sqluimpr****.*

Following is an example of the IMPORT command issued through the CLP:

```
db2 import from stafftab.ixf of ixf insert into userid.staff
```

To open the Import 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. Click the right mouse button on the table you want in the contents pane, and select Import from the pop-up menu. The Import notebook opens.

Detailed information about the Control Center is provided through its online help facility.

Related reference:

- "sqluimpr - Import" on page 67

Related samples:

- "tbmove.out -- HOW TO MOVE TABLE DATA (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.out -- HOW TO MOVE TABLE DATA (C++)"
- "tbmove.sqC -- How to move table data (C++)"

Using Import in a Client/Server Environment

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 will not 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 4KB.
- 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.

Using Import with Buffered Inserts

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

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
```

Note: In all environments except partitioned database environments, the buffered inserts feature is disabled during import operations in which the INSERT_UPDATE parameter is specified.

Using Import with Identity Columns

The import utility can be used to import data into a table containing an identity column. 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.

Two file type modifiers are supported by the import utility to simplify its use with tables that contain 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 may want to import data from a file (import.del) into TABLE1, and this data may 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 may 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
  ```

- 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 may 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 recreate 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).

Using Import with Generated Columns

The import utility can be used to import data into a table containing (non-identity) generated columns.

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

- A value will be 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.

Two file type modifiers are supported by the import utility to simplify its use with tables that contain 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 may want to import data from a file (load.del) into TABLE1, and this data may 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 may 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
```

- 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 may 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
```

Using Import to Recreate an Exported Table

You can use the import utility to recreate a table that was saved through the export utility. The table must have been exported to an IXF file, and the SELECT statement used during the export operation must have met certain conditions (for example, no column names can be used in the SELECT clause; only `select *` is permitted). When creating a table from an IXF file, not all attributes of the original table are preserved. For example, referential constraints, foreign key definitions, and user-defined data types are not retained. The following attributes of the original table are retained:

- Primary key name, and definition
- Unique constraints names, and definitions, but not other types of constraints or triggers
- Column information:
  - 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)
  - DATALINK options
  - 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:
  - Index name
  - Index creator name
  - Column names, and whether each column is sorted in ascending, or in descending order
  - Whether the index is defined as unique
  - Whether the index is clustered
  - Whether the index allows reverse scans
  - *pctfree* values
  - *minpctused* values

The following attributes of the original table are *not* retained:

- Whether the source was a normal table, a materialized query table, a view, or a set of columns from any or all of these sources
- Table information:
  - Materialized query table definition (if applicable)
  - Materialized query table options (if applicable)
  - Table space options; however, this information can be specified through the IMPORT command
- Column information:
  - Any default value except constant values
  - LOB options (if any)
  - References clause of the create table statement (if any)
  - Referential constraints (if any)

  - Check constraints (if any)
  - Generated column options (if any)
- Index information:
  - Include columns (if any)

Importing Large Objects (LOBS)

When importing into large object (LOB) columns, the data can come either from the same file as the rest of the column data, or from separate files. If the data is coming from separate files the LOBSINFILE file type modifier must be specified.

The column in the main input data file contains either the import data (default), or the name of a file where the import data is stored.

Notes:

1. When LOB data is stored in the main input data file, no more than 32KB of data is allowed. Truncation warnings are ignored.
2. All of the LOB data must be stored in the main file, or each LOB is stored in separate files. The main file cannot have a mixture of LOB data and file names. LOB values are imported from separate files by using the `lobsinfile` modifier, and the `LOBS FROM` clause.

A LOB Location Specifier (LLS) can be used to store multiple LOBs in a single file when importing, exporting and loading LOB information.

An LLS is a string indicating where LOB data can be found within a file. The format of the LLS is `filename.ext:nnn:mmm/`, where `filename.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 (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 256 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.

When importing or loading data with the `modified by lobsinfile` option specified, An LLS will be expected 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.

Related reference:

- “General Rules Governing PC/IXF File Import into Databases” on page 369

- “Data Type-Specific Rules Governing PC/IXF File Import into Databases” on page 371
- “IMPORT” on page 42

Importing User-defined Distinct Types (UDTs)

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.

Table Locking During Import

By default, the import utility is bound to the database with isolation level RR (repeatable read). If a large number of rows is being imported into a table, the existing lock may escalate to an exclusive lock. If another application working on the same table is holding some row locks, a deadlock will occur if the lock escalates to an exclusive lock. To avoid this, the import utility requests an exclusive lock on the table at the beginning of its operation.

Holding a lock on the 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 will wait until all of those applications commit or roll back their changes. Second, while import is running, any other application requesting locks will wait until the import operation has completed.

IMPORT

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

Authorization:

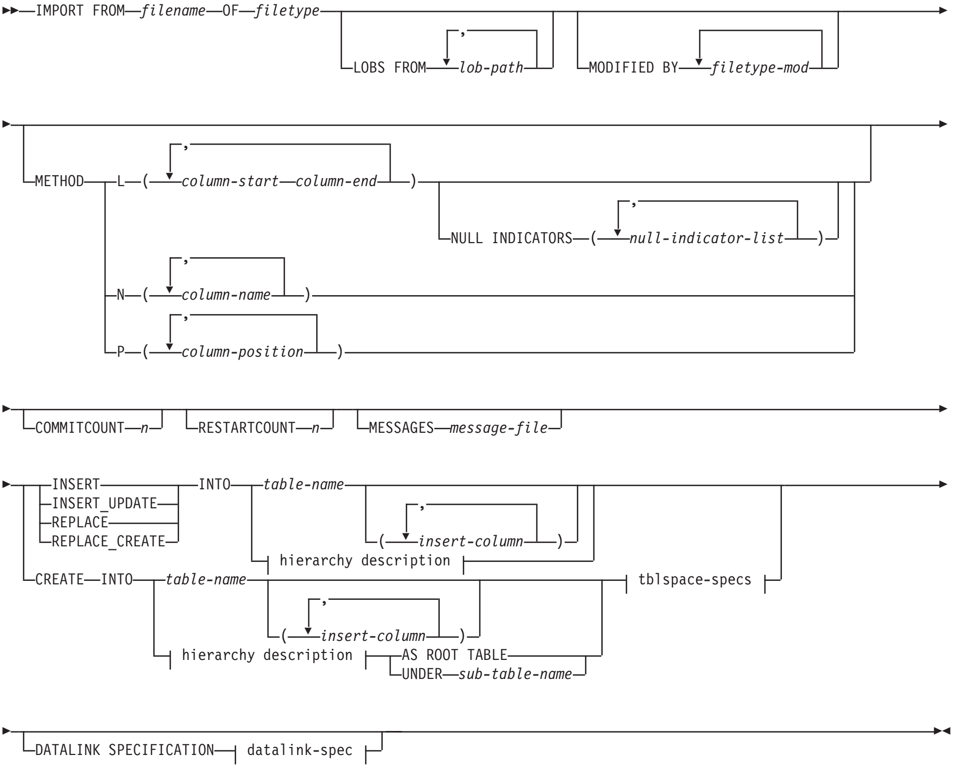
- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table or view
  - INSERT and SELECT privilege on each participating table or view.
- IMPORT to an existing table using the INSERT_UPDATE, REPLACE, or the REPLACE_CREATE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

  - *dbadm*
  - CONTROL privilege on the table or view.
- 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.

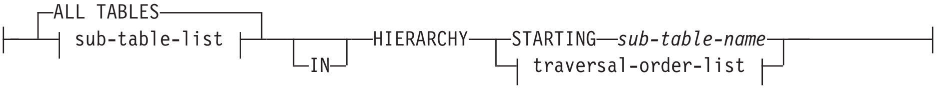
Required connection:

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

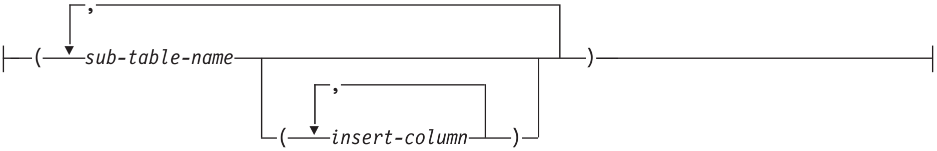
IMPORT



hierarchy description:



sub-table-list:



traversal-order-list:

```
        ┌─,──────────────┐
        ▼                │
├──(────sub-table-name───┴──)──────────────────────────────────────────────────┤
```

tblspace-specs:

```
├──┬────────────────────────────────────────────────────────────────────────────┬──┤
   └─IN──tablespace-name──┬─────────────────────────────┬──┬──────────────────────────┬─┘
                          └─INDEX IN──tablespace-name───┘  └─LONG IN──tablespace-name─┘
```

datalink-spec:

```
   ┌─,──────────────────────────────────────────────────────────────────────────────────────────┐
   ▼                                                                                            │
├──────(──┬─────────────────┬──┬──────────────────────────────────────┬──┬────────────────────────────┬──)─┴──┤
          └─DL_LINKTYPE URL─┘  ├─DL_URL_REPLACE_PREFIX──"prefix"──┤  └─DL_URL_SUFFIX──"suffix"──┘
                               └─DL_URL_DEFAULT_PREFIX──"prefix"──┘
```

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.

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

COMMITCOUNT n
Performs a COMMIT after every *n* records are imported.

CREATE
Creates the table definition and row contents. 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.

Note: If the data was exported from an MVS host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are less than 254, CREATE may fail because the rows are too long. In this case, the table should be created manually, and IMPORT with INSERT should be invoked, or, alternatively, the LOAD command should be used.

DATALINK SPECIFICATION
For each DATALINK column, there can be one column specification enclosed by parentheses. Each column specification consists of one or

more DL_LINKTYPE, prefix, and a DL_URL_SUFFIX specification. The prefix specification can be either DL_URL_REPLACE_PREFIX or DL_URL_DEFAULT_PREFIX.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns found within the *insert-column* list, or within the table definition (if an *insert-column* list is not specified).

DL_LINKTYPE

If specified, it should match the LINKTYPE of the column definition. Thus, DL_LINKTYPE URL is acceptable if the column definition specifies LINKTYPE URL.

DL_URL_DEFAULT_PREFIX ″prefix″

If specified, it should act as the default prefix for all DATALINK values within the same column. In this context, prefix refers to the ″scheme host port″ part of the URL specification. (For Distributed File System, or DFS, prefix refers to the ″scheme cellname filespace-junction″ part of the URL specification.)

Examples of prefix are:

```
"http://server"
"file://server"
"file:"
"http://server:80"
"dfs://.../cellname/fs"
```

If no prefix is found in a column’s data, and a default prefix is specified with DL_URL_DEFAULT_PREFIX, the default prefix is prefixed to the column value (if not NULL).

For example, if DL_URL_DEFAULT_PREFIX specifies the default prefix `"http://toronto"`:

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://coyote/a/b/c″.
- The column input value NULL is stored as NULL.

DL_URL_REPLACE_PREFIX ″prefix″

This clause is useful for loading or importing data previously generated by the export utility, when the user wants to globally replace the host name in the data with another host name. If specified, it becomes the prefix for *all* non-NULL column values. If a column value has a prefix, this will replace it. If a column value has no prefix, the prefix specified by DL_URL_REPLACE_PREFIX is prefixed to the column value. For Distributed File System (DFS), prefix refers to the ″scheme cellname filespace-junction″ part of the URL specification.

For example, if DL_URL_REPLACE_PREFIX specifies the prefix `"http://toronto"`:

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://toronto/a/b/c″. Note that ″toronto″ replaces ″coyote″.
- The column input value NULL is stored as NULL.

DL_URL_SUFFIX ″suffix″

If specified, it is appended to every non-NULL column value for the column. It is, in fact, appended to the ″path″ component of the URL part of the DATALINK value.

FROM filename

Specifies the file that contains the data to be imported. If the path is omitted, the current working directory is used.

HIERARCHY

Specifies that hierarchical data is to be imported.

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 a down-level 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

Specifies one or more paths that store LOB files. 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. This option is ignored if the `lobsinfile` modifier is not specified.

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.

MESSAGES message-file

Specifies the destination for warning and error messages that occur during an import operation. If the file already exists, the import utility appends the information. If the complete path to the file is not specified, the utility uses the current directory and the default drive as the destination. If *message-file* is omitted, the messages are written to standard output.

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 to be imported.

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

P Specifies the indexes (numbered from 1) 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 additional options (see Table 4 on page 56).

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 (see the description of the `nullindchar` modifier in Table 4 on page 56).

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 (integrated exchange format, PC version), which means it was exported from the same or another DB2 table. An IXF file also contains the table definition and definitions of any existing indexes, except when columns are specified in the SELECT statement.

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. It is not valid for tables with DATALINK columns. If this

option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

REPLACE_CREATE
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.

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

RESTARTCOUNT n
Specifies that an import operation is to be started at record *n* + 1. The first *n* records are skipped.

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.

Examples:

Example 1

The following example shows how to import information from `myfile.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 the table MOVIETABLE from the input file delfile1, which has data in the DEL format:

```
db2 import from delfile1 of del
   modified by dldel|
   insert into movietable (actorname, description, url_making_of,
   url_movie) datalink specification (dl_url_default_prefix
   "http://narang"), (dl_url_replace_prefix "http://bomdel"
   dl_url_suffix ".mpeg")
```

Notes:

1. The table has four columns:
   ```
   actorname              VARCHAR(n)
   description            VARCHAR(m)
   url_making_of          DATALINK (with LINKTYPE URL)
   url_movie              DATALINK (with LINKTYPE URL)
   ```
2. The DATALINK data in the input file has the vertical bar (|) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, ″http://narang″ is used.
4. Each non-NULL column value for url_movie will get ″http://bomdel″ as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ″.mpeg″ appended to the path. For example, if a column value of url_movie is ″http://server1/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″; if the value is ″/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″.

Example 3 (Importing into 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
```

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.

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. 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 60KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30KB of message information and the last 30KB 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 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. 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.

On the Windows NT operating system:

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

DB2 Data Links Manager Considerations

Before running the DB2 import utility, do the following:

1. Copy the files that will be referenced to the appropriate Data Links servers. The **dlfm_import** utility can be used to extract files from an archive that is generated by the **dlfm_export** utility.
2. Register the required prefix names to the DB2 Data Links Managers. There may be other administrative tasks, such as registering the database, if required.
3. Update the Data Links server information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required. (If the original configuration's Data Links servers are the same at the target

location, the Data Links server names need not be updated.) For Distributed File Systems (DFS), update the cell name information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required.

4. Define the Data Links servers at the target configuration in the DB2 Data Links Manager configuration file. For DFS, define the cells at the target configuration in the DB2 Data Links Manager configuration file.

When the import utility runs against the target database, files referred to by DATALINK column data are linked on the appropriate Data Links servers.

During the insert operation, DATALINK column processing links the files in the appropriate Data Links servers according to the column specifications at the target database.

Representation of DATALINK Information in an Input File

Table 4. Valid File Type Modifiers (Import)

Modifier	**Description**
	All File Formats
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.

Table 4. Valid File Type Modifiers (Import) (continued)

Modifier	Description
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. 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 4. Valid File Type Modifiers (Import) (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.
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: ",," is specified for the column • For ASC files: The NULL indicator is set to yes for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	

Table 4. Valid File Type Modifiers (Import) (continued)

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 to this code page from 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 ans x7F, inclusive. This refers to ASCII symbols and code points. **Notes:** 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 may be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[a] 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"`

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`.
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.

Table 4. Valid File Type Modifiers (Import) (continued)

Modifier	Description

timeformat=″*x*″	*x* is the format of the time in the source file.[a] 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 4. Valid File Type Modifiers (Import) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[a] 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 1 - 12; mutually exclusive with M, month)
	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)
	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 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 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`

ASC (Non-delimited ASCII) File Format	

Table 4. Valid File Type Modifiers (Import) (continued)

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.
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[b] 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.
	Note: This option replaces the obsolete `t` option, which is supported for back-level 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. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.
DEL (Delimited ASCII) File Format	

Table 4. Valid File Type Modifiers (Import) (continued)

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.[bc] 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.[bc]
	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`

datesiso	Date format. Causes all date data values to be imported in ISO format.
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.[bc]
	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"`

Table 4. Valid File Type Modifiers (Import) (continued)

Modifier	Description

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>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [bc] **Note:** *x* must not be the same character specified as the row, column, or character string 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.
nodoubledel	Suppresses recognition of double character delimiters.
IXF File Format	
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.

Table 4. Valid File Type Modifiers (Import) (continued)

Modifier	**Description**
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.
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.

Table 4. Valid File Type Modifiers (Import) (continued)

Modifier	Description

Notes:

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. [a] 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)
   ```

   Note: Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
3. [b] The character 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. [c] Delimiter Restrictions lists restrictions that apply to the characters that can be used as delimiter overrides.

Related reference:

- "db2look - DB2 Statistics and DDL Extraction Tool" in the *Command Reference*
- "LOAD" on page 133

sqluimpr - Import

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

Authorization:

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

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluimpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqluimpr (
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluimpt_in *pImportInfoIn,
    struct sqluimpt_out *pImportInfoOut,
    sqlint32 *pNullIndicators,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqluimpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgimpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluimpt_in *pImportInfoIn,
    struct sqluimpt_out *pImportInfoOut,
    sqlint32 *NullIndicators,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

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

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

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

pDataFileName
Input. A string containing the path and the name of the external input file from which the data is to be imported.

pLobPathList
Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA`, and the *sqlu_media_entry* structure listing paths on the client where the LOB files can be found.

pDataDescriptor
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 (defined in `sqlutil`) 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 *pDataDescriptor* 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. For ASC files the selection of columns from the external input files is to be read in from the file specified by the *pFileTypeMod* modified POSITIONSFILE.

pActionString

Input. Pointer to an *sqlchar* structure containing a 2-byte long field, followed by an array of characters identifying the columns into which data is to be imported.

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]}
[DATALINK SPECIFICATION datalink-spec]
```

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.

Attention: If an error occurs after the existing data is deleted, that data is lost.

CREATE
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.

REPLACE_CREATE
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.

Attention: If an error occurs after the existing data is deleted, that data is lost.

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 a down-level 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.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the IMPORT command.

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).

pFileType
Input. A string that indicates the format of the data within the external file. Supported external file 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 Integrated 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.

pFileTypeMod

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.

pMsgFileName

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.

CallerAction

Input. An action requested by the caller. Valid values (defined in `sqlutil`) 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.

pImportInfoIn
Input. Optional pointer to the *sqluimpt_in* structure containing additional input parameters.

pImportInfoOut
Output. Optional pointer to the *sqluimpt_out* structure containing additional output parameters.

NullIndicators
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 *pDataDescriptor* 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.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
IMPORT FROM datafile OF filetype
[MODIFIED BY :filetmod]
[METHOD {L|N|P} USING :dcoldata]
[COMMITCOUNT :commitcnt] [RESTARTCOUNT :restartcnt]
MESSAGES msgfile
{INSERT|REPLACE|CREATE|INSERT_UPDATE|REPLACE_CREATE}
INTO tname [(:columns)]
[OUTPUT INTO :output]

CONTINUE IMPORT

STOP IMPORT
```

REXX API parameter:

datafile
Name of the file from which the data is to be imported.

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

DEL Delimited ASCII

ASC Non-delimited ASCII

WSF Worksheet format

IXF PC version of Integrated Exchange Format.

filetmod
A host variable containing additional processing options.

L|N|P
A character specifying the method to be used to select columns within the external input file. Valid values are:

L Location

N Name

P Position.

dcoldata
A compound REXX host variable containing information about the columns selected for import from the external input file. The content of the structure depends upon the specified *method*. In the following, XXX represents the name of the host variable:

- Location method

  XXX.0 Number of elements in the remainder of the variable

  XXX.1 A number representing the starting location of this column in the input file. This column becomes the first column in the database table.

  XXX.2 A number representing the ending location of the column.

  XXX.3 A number representing the starting location of this column in the input file. This column becomes the second column in the database table.

  XXX.4 A number representing the ending location of the column.

  XXX.5 and so on.

- Name method

  XXX.0 Number of column names contained in the host variable.

  XXX.1 First name.

  XXX.2 Second name.

XXX.3 and so on.

- Position method

  XXX.0 Number of column positions contained in the host variable.

  XXX.1 A column position in the external input file.

  XXX.2 A column position in the external input file.

  XXX.3 and so on.

tname Name of the target table or view. Data cannot be imported to a read-only view.

columns
A REXX host variable containing the names of the columns in the table or the view into which the data is to be inserted. In the following, XXX represents the name of the host variable:

XXX.0 Number of columns.

XXX.1 First column name.

XXX.2 Second column name.

XXX.3 and so on.

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

commitcnt
Performs a COMMIT after every *commitcnt* records are imported.

restartcnt
Specifies that an import operation is to be started at record *restartcnt* + 1. The first *restartcnt* records are skipped.

output
A compound REXX host variable into which information from the import operation is passed. In the following, XXX represents the name of the host variable:

XXX.1 Number of records read from the external input file during the import operation.

XXX.2 Number of records skipped before inserting or updating begins.

XXX.3 Number of rows inserted into the target table.

XXX.4 Number of rows in the target table updated with information from the imported records.

XXX.5 Number of records that could not be imported.

XXX.6 Number of records imported successfully and committed to the database, including rows inserted, updated, skipped, and rejected.

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 *restartcnt* 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 *commitcnt* 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 60KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30KB of message information and the last 30KB 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. Non-default values for *pDataDescriptor*, or specifying an explicit list of table columns in *pActionString*, 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 NT operating system:

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

DB2 Data Links Manager Considerations

Before running the DB2 import utility, do the following:

1. Copy the files that will be referenced to the appropriate Data Links servers. The **dlfm_import** utility can be used to extract files from an archive that is generated by the **dlfm_export** utility.
2. Register the required prefix names to the DB2 Data Links Managers. There may be other administrative tasks, such as registering the database, if required.
3. Update the Data Links server information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required. (If the original configuration's Data Links servers are the same at the target location, the Data Links server names need not be updated.) For Distributed File Systems (DFS), update the cell name information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required.
4. Define the Data Links servers at the target configuration in the DB2 Data Links Manager configuration file. For DFS, define the cells at the target configuration in the DB2 Data Links Manager configuration file.

When the import utility runs against the target database, files referred to by DATALINK column data are linked on the appropriate Data Links servers.

During the insert operation, DATALINK column processing links the files in the appropriate Data Links servers according to the column specifications at the target database.

Representation of DATALINK Information in an Input File

Table 5. Valid File Type Modifiers (Import)

Modifier	Description
All File Formats	

Table 5. Valid File Type Modifiers (Import) (continued)

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 values.

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	Description
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.
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.
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: ",," is specified for the column • For ASC files: The NULL indicator is set to yes for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	

Table 5. Valid File Type Modifiers (Import) (continued)

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 to this code page from 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 ans x7F, inclusive. This refers to ASCII symbols and code points. **Notes:** 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 may be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[a] 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"`

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`.
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	Description

timeformat="*x*"	*x* is the format of the time in the source file.[a] 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 5. Valid File Type Modifiers (Import) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[a] 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 1 - 12; mutually exclusive with M, month)`
	`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)`
	`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 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 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`

ASC (Non-delimited ASCII) File Format	

Table 5. Valid File Type Modifiers (Import) (continued)

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.
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[b] 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.
	Note: This option replaces the obsolete `t` option, which is supported for back-level 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. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.
DEL (Delimited ASCII) File Format	

Table 5. Valid File Type Modifiers (Import) (continued)

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.[bc]
	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.[bc]
	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`

datesiso	Date format. Causes all date data values to be imported in ISO format.
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.[bc]
	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"`

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	Description

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>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [bc] **Note:** *x* must not be the same character specified as the row, column, or character string 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.
nodoubledel	Suppresses recognition of double character delimiters. For more information, see the "Delimiter restrictions" described in the **sqluexpr** API.
IXF File Format	
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.

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	**Description**
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.
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.

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	Description

Notes:

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. [a] 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)
   ```

   Note: Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
3. [b] The character 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. [c] The “Delimiter restrictions” described in the **sqluexpr** API list restrictions that apply to the characters that can be used as delimiter overrides.

Related reference:

- "SQLCA" in the *Administrative API Reference*
- "SQLU-MEDIA-LIST" in the *Administrative API Reference*

Related samples:

- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbmove.sqc -- How to move table data (C)"
- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tbmove.sqC -- How to move table data (C++)"

SQLUIMPT-IN

This structure is used to pass information to sqluimpr.

Table 6. Fields in the SQLUIMPT-IN Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
COMMITCNT	INTEGER	The number of records to import before committing them to the database. A COMMIT is performed whenever *commitcnt* records are imported.
RESTARTCNT	INTEGER	The number of records to skip before starting to insert or update records. This parameter should be used if a previous attempt to import records fails after some records have been committed to the database. The specified value represents a starting point for the next import operation.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-IN */
/* ... */
SQL_STRUCTURE sqluimpt_in
{
  sqluint32       sizeOfStruct;
  sqluint32       commitcnt;
  sqluint32       restartcnt;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-IN.
    05 SQL-SIZE-OF-UIMPT-IN   PIC 9(9) COMP-5 VALUE 12.
    05 SQL-COMMITCNT          PIC 9(9) COMP-5 VALUE 0.
    05 SQL-RESTARTCNT         PIC 9(9) COMP-5 VALUE 0.
*
```

Related reference:

- "sqluimpr - Import" on page 67

SQLUIMPT-OUT

This structure is used to pass information from sqluimpr.

Table 7. Fields in the SQLUIMPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
ROWSREAD	INTEGER	Number of records read from the file during import.
ROWSSKIPPED	INTEGER	Number of records skipped before inserting or updating begins.
ROWSINSERTED	INTEGER	Number of rows inserted into the target table.
ROWSUPDATED	INTEGER	Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).
ROWSREJECTED	INTEGER	Number of records that could not be imported.
ROWSCOMMITTED	INTEGER	The total number of processed records: the number of records inserted successfully and committed to the database, plus the number of skipped and rejected records.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-OUT */
/* ... */
SQL_STRUCTURE sqluimpt_out
{
  sqluint32       sizeOfStruct;
  sqluint32       rowsRead;
  sqluint32       rowsSkipped;
  sqluint32       rowsInserted;
  sqluint32       rowsUpdated;
  sqluint32       rowsRejected;
  sqluint32       rowsCommitted;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-OUT.
    05 SQL-SIZE-OF-UIMPT-OUT  PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWSREAD           PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSSKIPPED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSINSERTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSUPDATED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSREJECTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSCOMMITTED      PIC 9(9) COMP-5 VALUE 0.
*
```

Related reference:

- "sqluimpr - Import" on page 67

Character Set and NLS Considerations

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) may 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` modifier is specified, no initial comparison is performed, and an attempt is made to import the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is imported.

Related concepts:

- "Character Set and National Language Support" on page 235

Related reference:

- "IMPORT" on page 42

Import Sessions - CLP Examples

Example 1

The following example shows how to import information from `myfile.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 the table MOVIETABLE from the input file delfile1, which has data in the DEL format:

```
db2 import from delfile1 of del
    modified by dldel|
    insert into movietable (actorname, description, url_making_of,
    url_movie) datalink specification (dl_url_default_prefix
    "http://narang"), (dl_url_replace_prefix "http://bomdel"
    dl_url_suffix ".mpeg")
```

Notes:

1. The table has four columns:

```
actorname               VARCHAR(n)
description             VARCHAR(m)
url_making_of           DATALINK (with LINKTYPE URL)
url_movie               DATALINK (with LINKTYPE URL)
```

2. The DATALINK data in the input file has the vertical bar (|) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, ″http://narang″ is used.
4. Each non-NULL column value for url_movie will get ″http://bomdel″ as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ″.mpeg″ appended to the path. For example, if a column value of url_movie is ″http://server1/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″; if the value is ″/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″.

Example 3 (Importing into 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
```

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 4 (Importing Using 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)
```

Notes:

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.

Related concepts:

- “Nicknames and data source objects” in the *Federated Systems Guide*

Chapter 3. Load

This chapter describes the DB2 UDB load utility, which moves data from files, named pipes, devices or a cursor into a DB2 table. These data sources can reside either on the node where the database resides, or on a remotely connected client. The table being loaded must exist. If the table receiving the new data already contains data, you can replace or append to the existing data.

The following topics are covered:

- "Optimizing Load Performance" on page 237
- "Load - CLP Examples" on page 243

For information about loading DB2 Data Links Manager data, see "Using Import to Move DB2 Data Links Manager Data" on page 279.

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 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 1):

- Load, during which the data is written to the table.

  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. For DATALINK columns defined with FILE LINK CONTROL, link operations are performed for non-NULL column values. If a failure occurs, you can restart the load operation; the RESTART option automatically restarts the load operation from the last successful consistency point. The TERMINATE option rolls back the failed load operation.

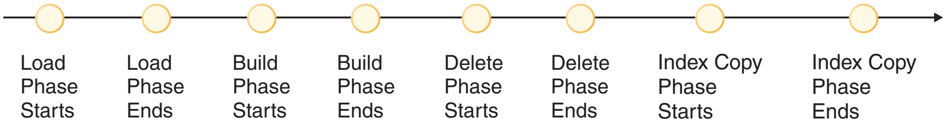


Figure 1. The Four Phases of the Load Process: Load, Build, Delete, and Index Copy. While the load operation is taking place, the target table is in the load in progress state. If the table has constraints, the table will also be in the check pending state. If the ALLOW READ ACCESS option was specified, the table will also be in the read access only state.

- Build, during which indexes are produced.

  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 YES with INDEXES option was specified). The statistics are similar to those collected through the RUNSTATS command. If a failure occurs during the build phase, the RESTART option automatically restarts the load operation at the appropriate point.

- Delete, during which the rows that caused a unique key violation or a DATALINK violation are removed from the table. Unique key violations are placed into the exception table, if one was specified, and messages about rejected rows are written to the message file. Following the completion of the load process, review these messages, resolve any problems, and insert corrected rows into the table.

  Do not attempt to delete or to modify any temporary files created by the load utility. Some temporary files are critical to the delete phase. If a failure occurs during the delete phase, the RESTART option automatically restarts the load operation at the appropriate point.

  Note: Each deletion event is logged. If you have a large number of records that violate the uniqueness condition, the log could fill up during the delete phase.
- Index copy, during which 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.

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.
- 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**.

You can also specify:

- That the data to be loaded resides on the client, if the load utility is invoked from a remotely connected client.
- 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. Use the SAVECOUNT parameter to specify this value. If this parameter is specified, a load restart operation will start at the last consistency point, instead of at the beginning.
- The names of the table columns into which the data is to be inserted.

- Whether or not pre-existing data in the table can be queried while the load operation is in progress.

  Note: This can be accomplished by using the READ ACCESS option and is not supported when the load utility is invoked in REPLACE mode.
- 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.

  Note: This is only supported when the READ ACCESS option is specified with a full index rebuild.
- The paths and the names of the input files in which LOBs are stored. The `lobsinfile` modifier tells the load utility that all LOB data is being loaded from files.
- A message file name. During DB2® operations such as exporting, importing, loading, binding, or restoring data, 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. These message files are standard ASCII text files. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

  Notes:

  1. You can only view the contents of a message file after the operation is finished.
  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. The `implieddecimal` modifier tells the load utility that decimal points are to be applied to the data as it enters the table. For example, the value 12345 is loaded into a DECIMAL(8,2) column as 123.45, not 12345.00.
- Whether the utility should modify the amount of free space available after a table is loaded. Additional free space permits INSERT and UPDATE growth to the table following the completion of a load operation. Reduced free space keeps related rows more closely together and can enhance table performance.

  Note: For multi-dimensional clustering (MDC) tables, the `totalfreespace`, modifier is not supported.
  - The `totalfreespace` modifier enables you to append empty data pages to the end of the loaded table. The number specified with this parameter is 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 you specified the

  number twenty with this parameter, and the table has 100 data pages, twenty additional empty pages are appended. The total number of data pages in the table will then be 120.
  - The pagefreespace modifier enables you to control the amount of free space that will be allowed on each loaded data page. The number specified with this parameter is the percentage of each data page that is to be left as free space. The first row in a page is added without restriction. Therefore, with very large rows and a large number specified with this parameter, there may be less free space left on each page than that indicated by the value specified with this parameter.
  - The indexfreespace modifier enables you to control the amount of free space that will be allowed on each loaded index page. The number specified with this parameter is the percentage of each index page that is to be left as free space. The first index entry in a page is added without restriction. Additional index entries are placed in the index page, provided the percent free space threshold can be maintained. The default value is the one used at CREATE INDEX time. The indexfreespace value takes precedence over the PCTFREE value specified in the CREATE INDEX statement.

  If you specify the pagefreespace modifier, and you have an index on the table, you might consider specifying indexfreespace. When deciding on the amount of free space to leave for each, consider that the size of each row being inserted into the table will likely be larger than the size of the associated key to be inserted into the index. In addition, the page size of the table spaces for the table and the index may be different.

- Whether statistics are to be gathered during the load process. This option is only supported if the load operation is running in REPLACE mode.

  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 forward log recovery is disabled for the database; that is, if the database configuration parameters *logretain* and *userexit* are disabled. If no copy is made, and forward log 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 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 2). 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.

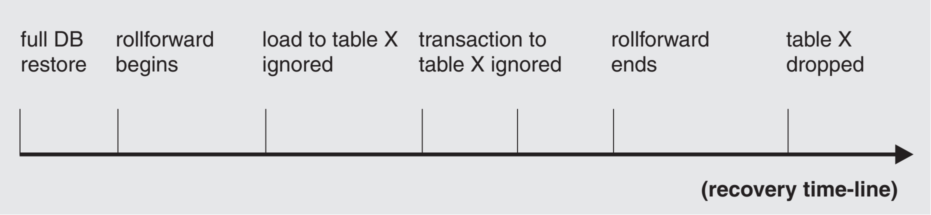


Figure 2. Non-recoverable Processing During a Roll Forward Operation

- 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. This is true even if you are the instance owner. If you are not the instance owner, you must specify a location that is writable by the instance owner.

Changes to Previous Load Behavior Introduced in Version 6 and Version 7

The Version 6 and the Version 7 load utilities have full back-level compatibility with previous releases; that is, they will accept syntax from previous releases and operate normally. Following is a summary of syntax changes and changes to load behavior introduced in Version 6 and Version 7:

- Load restart no longer uses a RESTARTCOUNT value, and the parameter is now reserved. When a previously interrupted load operation is restarted, the load operation will automatically continue from the last consistency point in the load, build, or delete phase.
- The sorting of index keys during index creation takes advantage of a new sorting algorithm used by the DB2 UDB Version 6 database engine. The amount of memory dedicated to sorting is controlled by the values of the sort heap (*sortheap*) database configuration parameter, and the sort heap threshold (*sheapthres*) database manager configuration parameter. If these options are specified in Version 6, an informational warning message is returned, but the load operation proceeds normally.

  Sort spills that occur during load index creation are now performed inside a temporary table space. Sort operations do not spill directly to disk; rather, they spill to the buffer pool associated with the temporary table space. Having a large buffer pool associated with the temporary table space can improve index creation time. To achieve the same type of I/O parallelism that was available in pre-Version 6 load sort operations (by specifying multiple temporary sort directories), it is recommended that temporary table spaces be declared with multiple containers, each residing on a different disk device. It is also recommended that temporary table spaces be declared as SMS (System Managed Space), so that they can grow to accommodate large volumes of data without holding disk resources when those resources are not being used.
- The REMOTE FILE option has been renamed (although the utility still accepts the REMOTE FILE parameter when specifying the path to temporary files). This is a purely syntactical change to better reflect the meaning and purpose of this parameter. The TEMPFILES PATH parameter refers to a directory, rather than a file.
- The load utility now supports several indexing modes (full REBUILD, INCREMENTAL extensions, index maintenance DEFERRED until after a load operation completes, and an AUTOSELECTion mode, which chooses between full rebuild and incremental maintenance at run time). The full rebuild mode mirrors the behavior of pre-Version 6 releases. The default behavior for Version 6 is AUTOSELECT mode.
- In Version 6, the TERMINATE option can be used to roll back a load operation. Previously this option would put a table space in restore pending state. Note, however, that a TERMINATE request after a failed LOAD REPLACE operation will *not* restore the table data.

The Version 7 load utility can load data residing on a remotely connected client, in fully qualified files or named pipes. (Separate files containing LOB values, when the `lobsinfile` file type modifier is specified, should be located on the server.)

Changes to Previous Load Behavior Introduced in Version 8

Following is a summary of syntax changes and changes to load behavior introduced in Version 8:

- Prior to Version 8, load required exclusive access to table spaces that contained objects belonging to the table being loaded. In Version 8, load operates at the table level and no longer requires exclusive access to the table space. Load will place a lock only on the table objects associated with the load operation taking place. Concurrent access to other table objects in the same table spaces is permitted.

  Note: Prior to Version 8, when the COPY NO option was specified on a recoverable database, the table space was put in backup pending state only after the load operation was committed. In Version 8, the table space will be placed in backup pending state when the load operation begins and will remain in that state even if the load operation fails and is rolled back. As in previous releases, when the COPY NO option is specified and load operation completes successfully, the rollforward utility will put dependent table spaces in restore pending state during a rollforward operation.
- You can also specify that users have read access to the data that existed in the table prior to the load. This means that after the load operation has completed, you will not be able to view the new data if there are constraints on the table and integrity checking has not been completed. You can also specify that the index be rebuilt in a separate table space during a load operation by specifying the READ ACCESS and INDEXING MODE REBUILD options. The index will be copied back to the original table space during the index copy phase which occurs after the other phases of the load operation.
- The functionality of the LOAD QUERY command has been expanded and it now returns the table state of the target into which data is being loaded in addition to the status information it previously included on a load operation in progress. The LOAD QUERY command may also be used to query the table state whether or not a load operation is in progress on that table.
- Extent allocations in DMS table spaces are now logged. The LOAD command will now write two log records for every extent it allocates in a DMS table space. Also, when the READ ACCESS and INDEXING MODE INCREMENTAL options are specified, some log records will be written while data is being incrementally inserted into the index.

- Dependent table spaces will no longer be quiesced prior to a load operation. When the COPY NO option is specified, the new table space state *load in progress* will be used. 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 in that all load operations use the load in progress table state, but load operations with the COPY NO option specified also use the load in progress table space state.
- When executing a load operation with the ALLOW READ ACCESS and INDEXING MODE REBUILD options, a new copy of the indexes is created in addition to the original indexes. This means that the space requirement for the index table space may have to be doubled. To avoid this, the USE TABLESPACE option can be used to specify a temporary table space for the storage of new indexes. After the new indexes are built in the temporary table space, the target table is taken offline before the new indexes are copied into the target table space.
- Calls to quiesce table spaces from the LOAD command have been removed. If you quiesce table spaces in exclusive mode prior to a load operation, you will now have to explicitly remove the table spaces from the quiesced exclusive state. In previous releases, after issuing the following commands LOAD would have implicitly reset the quiesced table spaces and made them accessible to other applications:

  ```
  quiesce tablespaces for table t1 exclusive
  load from data.del of del insert into t1
  ```

  In Version 8, you must issue the following command to remove the table space from the quiesced exclusive state:

  ```
  quiesce tablespaces for table t1 reset
  ```
- A LOCK WITH FORCE option has been added to the LOAD command. It allows you to force other applications to release locks they have on a table and to allow the load operation to proceed and acquire the locks it needs.
- The load utility now has the ability to load from an SQL statement, using the new CURSOR file type.
- Loading data that resides on a remotely connected client is now supported under the following conditions:
  - The database that the client is connected to is in a partitioned database environment.
  - The database that the client is connected to is cataloged against an already cataloged database.
- Loading data into multi-dimensional clustering (MDC) tables is supported.
- Prior to Version 8, following a load operation the target table remained in check pending state if it contained generated columns. The load utility will

now generate column values, and you are no longer required issue the SET INTEGRITY statement after a load operation.

- Tables can now be loaded in a partitioned database environment. The AutoLoader utility (**db2atld**) is no longer required to accomplish this. The load API (**db2Load**) has also been enhanced to support the partitioned database load options.

Related concepts:

- "Rollforward Recovery" in the *Data Recovery and High Availability Guide and Reference*

Related reference:

- "RUNSTATS" in the *Command Reference*
- "LOAD QUERY" on page 178
- "LOAD" on page 133
- "db2Load - Load" on page 181

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 3). Another way involves the use of multiple processors in an SMP environment, which allows for intra-partition parallelism (see Figure 4 on page 111). Both can be used together to provide even faster loading of data.

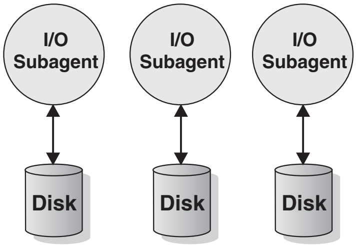


Figure 3. Taking Advantage of I/O Parallelism When Loading Data

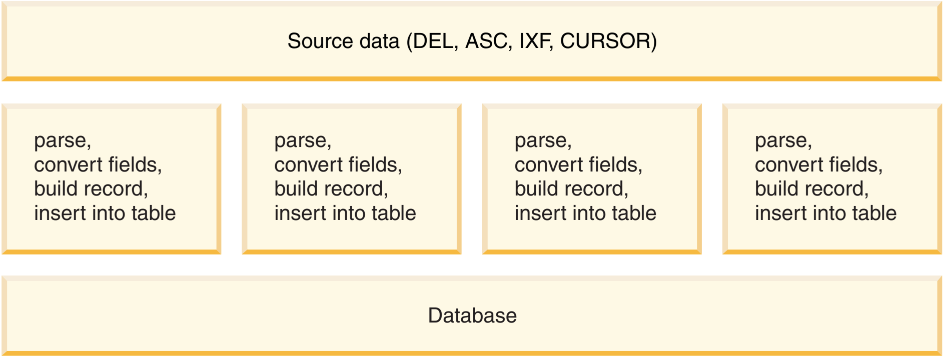

Source data (DEL, ASC, IXF, CURSOR)
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
Database

Figure 4. Taking Advantage of Intra-partition Parallelism When Loading Data

Related concepts:

- "Optimizing Load Performance" on page 237

Privileges, Authorities, and Authorizations Required to Use Load

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.

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.

On Windows® NT/2000/XP 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.

Related reference:

- "LOAD" on page 133
- "db2Load - Load" on page 181

Using Load

Prerequisites:

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 performing either a COMMIT or a ROLLBACK before invoking load.

Since data is loaded in the sequence that appears in the input file (except when using multi-dimensional clustering (MDC) tables), if a particular sequence is desired, the data should be sorted before a load operation is attempted.

If clustering is required, the data should be sorted on the clustering index prior to loading. When loading data into MDC tables, sorting is not required prior to the load operation, and data will be clustered according to the MDC table definition.

Restrictions:

The following restrictions apply to the load utility:

- 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.
- Attempts to create or drop tables in a table space that is in backup pending state will fail.
- You cannot load data into a database accessed through DB2 Connect or a down-level server prior to DB2 Version 2. Options that are only available with this release of DB2 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.

Procedure:

The load utility can be invoked through the command line processor (CLP), the Load notebook in the Control Centre, or an application programming interface (API), **db2Load**.

Following is an example of the 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, formerly ADSM).
- Four thousand pages of buffer space are to be used during the load operation.

Following is another example of the 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 node as the database. The use of fully qualified path names is recommended.

To open the Load 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. Click the right mouse button on the table you want in the contents pane, and select Load from the pop-up menu. The Load notebook opens.

Detailed information about the Control Center is provided through its online help facility.

Related reference:

- "Tivoli Storage Manager" in the *Data Recovery and High Availability Guide and Reference*

Related samples:

- "tbmove.out -- HOW TO MOVE TABLE DATA (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.out -- HOW TO MOVE TABLE DATA (C++)"
- "tbmove.sqC -- How to move table data (C++)"

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. This is the default behavior. The ALLOW READ ACCESS option prevents all write access to the table by other applications, but allows read access to pre-loaded 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 pre-existing data, but they will not be able to read the newly loaded data until the load operation has finished.

Read access is provided throughout the load operation except at the very end. Before data is committed the load utility acquires an exclusive lock (Z-lock) on the table. The load utility will wait until all applications that have locks on the table, release them. This may cause a delay before the data can be committed. The LOCK WITH FORCE option may be 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 <tablespaceName> option is specified, the exclusive lock will last for the entire period of the index copy phase.

Notes:

1. If a load operation is aborted, it remains at the same access level that was specified when the load operation was issued. So, if a load operation in ALLOW NO ACCESS mode aborts, 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 pre-loaded table data is still accessible for read access.
2. If the ALLOW READ ACCESS option was specified for an aborted load operation, it can also be specified for the load restart or load terminate operation. However, if the aborted 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 pre-existing 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 check pending state and is not in read 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.

Related concepts:

- "Checking for Integrity Violations" on page 123
- "Table Locking, Table States and Table Space States" on page 231
- "Building Indexes" on page 117

Building Indexes

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 will be rebuilt.
2. INCREMENTAL. Indexes will be extended with new data.
3. AUTOSELECT. The load utility will automatically decide between REBUILD or INCREMENTAL mode. This is the default.

   Note: You may decide to explicitly choose an indexing mode because the behavior of the REBUILD and INCREMENTAL modes are quite different.
4. DEFERRED. The load utility will not attempt index creation if this mode is specified. Indexes will be marked as needing a refresh, and a rebuild may be forced the first time they are accessed. This option is not compatible with the ALLOW READ ACCESS option because it does not maintain the indexes and index scanners require a valid index.

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 will write 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.

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 will remain intact.

Building New Indexes in the Same Table Space as the Original

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 will take place almost instantaneously.

If the indexes are built in a DMS table space, the new *shadow* index cannot be seen by the user. If the indexes are built within an SMS table space, the user may 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.

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 <tablespaceName> 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 may 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 <tablespaceName> option is ignored if the load operation is not in ALLOW READ ACCESS mode, or if the indexing mode is incompatible. The USE <tablespaceName> 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 <tablespaceName> option will be ignored.

A load restart operation may 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 <tablespaceName> option will be 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 is going to run out of index space, it will do it during the build phase. If this happens, the original index will not be 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 may 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 will be used.

Using Load with Identity Columns

The load utility can be used to load data into a table containing an identity column. 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 will not be reported.

Three (mutually exclusive) file type modifiers are supported by the load utility to simplify its use with tables that contain an identity column:

- 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))
```

  A user may 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. The following is an example of such a file:

```
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 may 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
```

- The `identityignore` modifier is in some ways the opposite of the `identitymissing` modifier: it 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 to be used for the identity column, the user could 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 may 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
  ```

- 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).

  Note: When using this modifier, it is possible to violate the uniqueness property of GENERATED ALWAYS columns.

Using Load with Generated Columns

The load utility can be used to load data into a table containing (non-identity) generated columns. The column values will be generated by this utility. This means that following a load operation, the target table will not be left in check pending state if the only constraints on the table are generated column constraints.

Note: If you initiate a load operation between a Version 7 or earlier client and a Version 8 or later server, the load utility will place tables with generated columns in check pending state.

If a table has been placed in check pending state because a Version 7 or earlier client was used to load data into a table with generated columns, the following statement will take the table out of check pending state and force the generation of values:

```
SET INTEGRITY FOR tablename IMMEDIATE CHECKED FORCE GENERATED;
```

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

- Values will be 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 will be 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.

Two (mutually exclusive) file type modifiers are supported by the load utility to simplify its use with tables that contain 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))
  ```

  A user may 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. The following is an example of such a file:

```
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 may 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
```

- The generatedignore modifier is in some ways the opposite of the generatedmissing modifier: it 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, a user might 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). 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 may 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
  ```

- The generatedoverride modifier is used for loading user-supplied values into a table with generated columns. This can be quite 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).

Checking for Integrity Violations

Following a load operation, the loaded table may be in check 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 datalink columns defined on it.
- The table has generated columns and a Version 7 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 check 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 CHECK PENDING CASCADE parameter can be specified to indicate whether or not the check 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 CHECK PENDING CASCADE IMMEDIATE

The loaded table, its dependent materialized query tables and dependent staging tables will be placed in check pending state with read access.

INSERT, ALLOW READ ACCESS, and CHECK PENDING CASCADE DEFERRED

Only the loaded table will be placed in check pending with read access. Descendent foreign key tables, descendent materialized query tables and descendent staging tables will remain in their original states.

INSERT, ALLOW NO ACCESS, and CHECK PENDING CASCADE IMMEDIATE

The loaded table, its dependent materialized query tables and dependent staging tables will be placed in check pending state with no access.

INSERT or REPLACE, ALLOW NO ACCESS, and CHECK PENDING CASCADE DEFERRED

Only the loaded table will be placed in check pending state with no access. Descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain in their original states.

REPLACE, ALLOW NO ACCESS, and CHECK PENDING CASCADE IMMEDIATE

The table and all its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be placed in check pending state with no access.

Note: Specifying the ALLOW READ ACCESS option in a load replace operation will result in an error.

To remove the check pending state, use the SET INTEGRITY statement. The SET INTEGRITY statement checks a table for constraints violations, and takes the table out of check 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 will check 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 CHECK PENDING CASCADE DEFERRED option specified, and the SET INTEGRITY statement is used to check for integrity violations, the descendent tables will be placed in check 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 will be taken out of check pending state and placed in NO DATA MOVEMENT mode. 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, operations that may cause the movement of rows within the table are not allowed.

You can override the NO DATA MOVEMENT mode by specifying the FULL ACCESS option when you issue the SET INTEGRITY statement. The table will be fully accessible, however a full recomputation of the dependent materialized query tables will take place in subsequent REFRESH TABLE statements and the dependent staging tables will be forced into an incomplete state.

If the ALLOW READ ACCESS option is specified for a load operation, the table will remain in read access state until the SET INTEGRITY statement is used to check for constraints violations. Applications will be able to 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 has been 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 it existed in the table prior to the first load operation will be 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 check 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 may 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 DB2® will select 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 were added to the table while it was in check pending state.
- A load replace operation has taken place, or the NOT LOGGED INITIALLY WITH EMPTY TABLE option was activated, after the last integrity check on the table.
- A parent table has been load replaced or checked for integrity non-incrementally.
- The table was in check 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 has been 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 will be incrementally processed and the CONST_CHECKED column of SYSCAT.TABLES will be marked as `U` to indicate that not all data has been verified by the system.

Use the load exception table option to capture information about rows with constraints violations.

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 check pending state, triggers are active. Thus, if we correct data and insert rows from the exception table into the loaded table, any INSERT triggers defined on the table will be 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.

Related reference:

- "SET INTEGRITY statement" in the *SQL Reference, Volume 2*

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 will result 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) will be 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 check 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 check 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 will be incrementally refreshed and the CONST_CHECKED column of SYSCAT.TABLES will be 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 will be checked for data integrity and will be placed in no data movement mode. UT1 will be 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 will be 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;
```

Related concepts:

- "Checking for Integrity Violations" on page 123

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 will be 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 will also be 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 will be marked incomplete. That is, the staging byte of the CONST_CHECKED column will have 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 will be 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 will turn off immediate data propagation and set 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 check 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 will be 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 will be 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;
```

Related concepts:

- “Checking for Integrity Violations” on page 123

Multi-Dimensional Clustering Considerations

The following restrictions apply to multi-dimensional 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.

When using the LOAD command with MDC, violations of unique constraints will 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 will remain and the new records will be 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 will be loaded and the others will be deleted during the delete phase.

  Note: There is no explicit technique for determining which record will be loaded and which will be deleted.

Performance Considerations

To improve the performance of the load utility when loading MDC tables, the UTIL_HEAP_SZ database configuration parameter should be set to a value that is 10-15% higher than usual. This will reduce disk I/O during the clustering of data that is performed during the load phase. When the DATA BUFFER option of LOAD command is specified, its value should also be increased by 10-15%. If the LOAD command is being used to load several MDC tables concurrently, the UTIL_HEAP_SZ configuration parameter should be increased accordingly.

MDC load operations will 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 will be 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.

Related concepts:

- "Optimizing Load Performance" on page 237

Restarting an Interrupted Load Operation

If the load utility cannot start because of a user error, such as a nonexistent data file or invalid column names, it will terminate and leave the table in a normal state.

If a failure occurs while loading data, you can restart the load operation from the last consistency point (using the RESTART option), or reload the entire table (using the REPLACE option). Specify the same parameters as in the previous invocation, so that the utility can find the necessary temporary files. Because the SAVECOUNT parameter is not supported for multi-dimensional clustering (MDC) tables, a load restart will only take place at the beginning of the load, build, or delete phase.

Note: A load operation that specified the ALLOW READ ACCESS option can be restarted using either the ALLOW READ ACCESS option or the ALLOW NO ACCESS option. Conversely, a load operation that specified the ALLOW NO ACCESS option can not be restarted using the ALLOW READ ACCESS option.

Restarting or Terminating an Allow Read Access Load Operation

A aborted load operation that specified the ALLOW READ ACCESS option may also be restarted or terminated using the ALLOW READ ACCESS option. This will allow 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 will not be permitted.

If the original load operation was aborted in the index copy phase, a restart operation in the ALLOW READ ACCESS mode is not permitted because the index may be corrupted.

If a load operation in ALLOW READ ACCESS mode was aborted in the load phase, it will restart in the load phase. If it was aborted in any phase other than the load phase, it will restart in the build phase. If the original load operation was in ALLOW NO ACCESS mode, a restart operation may occur in the delete phase if the original load operation reached that point and the index is valid. If the index is marked invalid, the load utility will restart the load operation from the build phase.

Note: All load restart operations will choose the REBUILD indexing mode even if the INDEXING MODE INCREMENTAL option is specified.

Issuing a LOAD TERMINATE command will generally cause the aborted 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 may be a delay while the load utility scans the indexes and corrects any inconsistencies. The length of this delay will depend on the size of the indexes and will occur whether or not the ALLOW READ ACCESS option is specified for the load terminate operation. The delay will not occur if the original load operation failed prior to the build phase.

Note: The delay resulting from corrections to inconsistencies in the index will be 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. The 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 aborted load operation but do not roll forward to the end of the load terminate or load restart operation.

Related concepts:

- "Table Locking, Table States and Table Space States" on page 231
- "Restarting or Terminating a Load Operation in a Partitioned Database Environment" on page 260

Related reference:

- "Partitioned Database Load Configuration Options" on page 262

Using 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. It 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 may be overwritten before rollforward recovery takes place.

Notes:

1. In a partitioned database environment, the DB2LOADREC registry variable must be set for all the database partition servers using the **db2set** command.
2. In a partitioned database environment, 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
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
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2®            TORONTO
SES            NULL
BUF            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
```

```
BUF          NULL
TYP          O
SHRlib       /@sys/lib/backup_vendor.a
```

Notes:

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. If SES is NULL, the value specified by the *numloadrecses* 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 rebuild 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 partitioned database environment, the recovery history file is local to each database partition.

Related reference:

- "Tivoli Storage Manager" in the *Data Recovery and High Availability Guide and Reference*

LOAD

Loads data into a DB2 table. Data residing on the server may be in the form of a file, tape, or named pipe. Data residing on a remotely connected client may be in the form of a fully qualified file or named pipe. Data can also be loaded from a user defined cursor. The load utility does not support loading data at the hierarchy level.

Scope:

This command may 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.

Note: 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.

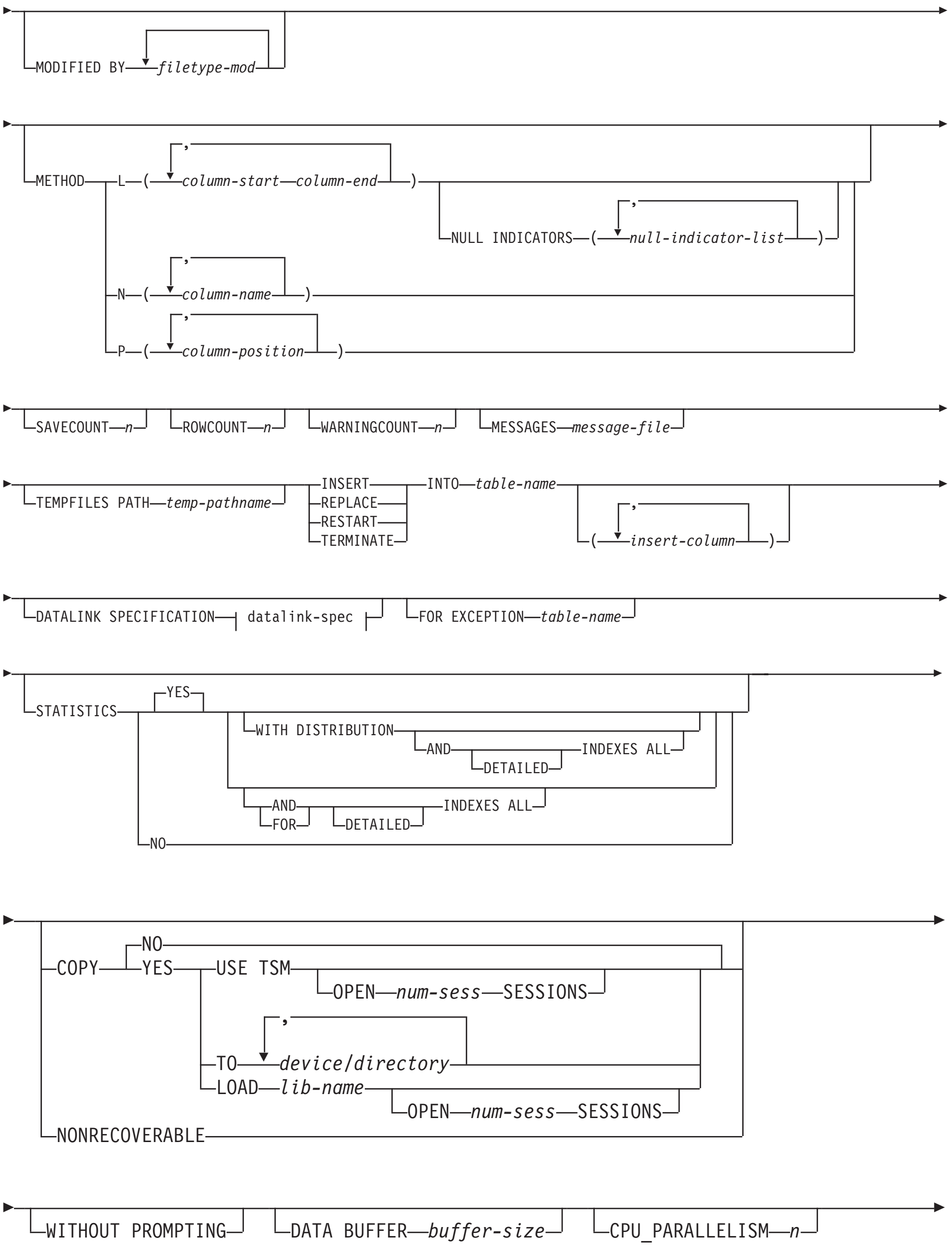
Required connection:

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

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:

```
                          ┌─,──────────────┐
                          ▼                │
►►──LOAD──┬────────┬──FROM────┬─filename───┬─┴──OF──filetype──┬──────────────────────────┬──────────►
          └─CLIENT─┘          ├─pipename───┤                  │             ┌─,────────┐ │
                              ├─device─────┤                  │             ▼          │ │
                              └─cursorname─┘                  └─LOBS FROM──────lob-path─┴─┘
```


MODIFIED BY
filetype-mod
METHOD
L
(
,
column-start
column-end
)
NULL INDICATORS
(
,
null-indicator-list
)
N
(
,
column-name
)
P
(
,
column-position
)
SAVECOUNT
n
ROWCOUNT
n
WARNINGCOUNT
n
MESSAGES
message-file
TEMPFILES PATH
temp-pathname
INSERT
REPLACE
RESTART
TERMINATE
INTO
table-name
(
,
insert-column
)
DATALINK SPECIFICATION
datalink-spec
FOR EXCEPTION
table-name
STATISTICS
YES
WITH DISTRIBUTION
AND
DETAILED
INDEXES ALL
AND
FOR
DETAILED
INDEXES ALL
NO
COPY
NO
YES
USE TSM
OPEN
num-sess
SESSIONS
TO
,
device/directory
LOAD
lib-name
OPEN
num-sess
SESSIONS
NONRECOVERABLE
WITHOUT PROMPTING
DATA BUFFER
buffer-size
CPU_PARALLELISM
n

```
►─┬─────────────────────┬──┬─────────────────────────────────────────┬──────────────────►
  └─DISK_PARALLELISM──n─┘  └─INDEXING MODE──┬─AUTOSELECT──┬─┘
                                              ├─REBUILD─────┤
                                              ├─INCREMENTAL─┤
                                              └─DEFERRED────┘

   ┌─ALLOW NO ACCESS─────────────────────────────┐
►──┼──────────────────────────────────────────────┼──────────────────────────────────────►
   └─ALLOW READ ACCESS──┬──────────────────────┬──┘
                        └─USE──tablespace-name─┘

►─┬────────────────────────────────────────────────┬──┬─────────────────┬────────────────►
  └─CHECK PENDING CASCADE──┬─IMMEDIATE─┬─┘          └─LOCK WITH FORCE─┘
                           └─DEFERRED──┘

►─┬─────────────────────────────────────────────────────────────────────────────────┬──►◄
  │                                                   ┌─────────────────────────┐  │
  │  ┌─PARTITIONED DB CONFIG─┐                         ▼                         │  │
  └──┴───────────────────────┴─partitioned-db-option───partitioned-db-option──┴──┘
```

datalink-spec:

```
     ┌─,──────────────────────────────────────────────────────────────────────────┐
     ▼                                                                            │
├────(─┬───────────────────┬──┬──────────────────────────────────────┬──┬──────────────────────────┬─)─┴───────┤
       └─DL_LINKTYPE URL───┘  ├─DL_URL_REPLACE_PREFIX──"prefix"─────┤  └─DL_URL_SUFFIX──"suffix"──┘
                              └─DL_URL_DEFAULT_PREFIX──"prefix"─────┘
```

Command parameters:

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 behaviour. It is the the only valid option for LOAD REPLACE.

When there are constraints on the table, the table state will be set to CHECK PENDING as well as LOAD IN PROGRESS. The SET INTEGRITY command must be used to take the table out of CHECK PENDING.

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 may 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 may use this option; LOAD TERMINATE or LOAD RESTART of an ALLOW NO ACCESS

load may not 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 CHECK PENDING as well as LOAD IN PROGRESS, and READ ACCESS. At the end of the load the table state LOAD IN PROGRESS will be unset but the table states CHECK PENDING and READ ACCESS will remain. The SET INTEGRITY command must be used to take the table out of CHECK PENDING. While the table is in CHECK PENDING and READ ACCESS, 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 command has completed. A user may perform multiple loads on the same table without issuing a SET INTEGRITY command. Only the original (checked) data will remain visible, however, until the SET INTEGRITY command 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 tablespace 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 may 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.

CHECK PENDING CASCADE

If LOAD puts the table into a check pending state, the CHECK

PENDING CASCADE option allows the user to specify whether or not the check 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 the check pending state (read or no access mode) for foreign key constraints is immediately extended to all descendent foreign key tables. If the table has descendent immediate materialized query tables or descendent immediate staging tables, the check pending state is extended immediately to the materialized query tables and the staging tables. Note that for a LOAD INSERT operation, the check 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 check pending read state will be put into check pending no access state.

DEFERRED

Indicates that only the loaded table will be placed in the check pending state (read or no access mode). 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 may later be implicitly placed in the check pending no access state when their parent tables are checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statment). Descendent immediate materialized query tables and descendent immediate staging tables will be implicitly placed in the check pending no access 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 the check pending state. See the Notes section of the SET INTEGRITY statement in the SQL Reference for when these descendent tables will be put into the check pending state.

If the CHECK PENDING CASCADE option is not specified:

- Only the loaded table will be placed in the check pending state. The state of descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged, and may later be implicitly put into the check pending state when the loaded table is checked for constraint violations.

If LOAD does not put the target table into check pending state, the CHECK PENDING CASCADE option is ignored.

CLIENT

Specifies that the data to be loaded resides on a remotely connected client. This option is ignored if the load operation is not being invoked from a remote client. This option is not supported in conjunction with the CURSOR filetype.

Notes:

1. The DUMPFILE and LOBSINFILE modifiers refer to files on the server even when the CLIENT keyword is specified.
2. Code page conversion is not performed during a remote load operation. If the code page of the data is different from that of the server, the data code page should be specified using the CODEPAGE modifier.

In the following example, a data file (`/u/user/data.del`) residing on a remotely connected client is to be loaded into MYTABLE on the server database:

```
db2 load client from /u/user/data.del of del
   modified by codepage=850 insert into mytable
```

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). Copy no, 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.

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). The option is not supported for tables with DATALINK columns.

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 may contain the full path. If the full path is not given, it will default to the path where the user exit programs reside.

CPU_PARALLELISM n
Specifies the number of processes or threads that the load utility will spawn 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.

Notes:

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.

DATA BUFFER buffer-size
Specifies 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 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.

DATALINK SPECIFICATION

For each DATALINK column, there can be one column specification enclosed by parentheses. Each column specification consists of one or more DL_LINKTYPE, prefix, and a DL_URL_SUFFIX specification. The prefix specification can be either DL_URL_REPLACE_PREFIX or DL_URL_DEFAULT_PREFIX.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns found within the *insert-column* list, or within the table definition (if an *insert-column* list is not specified).

DISK_PARALLELISM n

Specifies the number of processes or threads that the load utility will spawn 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.

DL_LINKTYPE

If specified, it should match the LINKTYPE of the column definition. Thus, DL_LINKTYPE URL is acceptable if the column definition specifies LINKTYPE URL.

DL_URL_DEFAULT_PREFIX ″prefix″

If specified, it should act as the default prefix for all DATALINK values within the same column. In this context, prefix refers to the ″scheme host port″ part of the URL specification. (For Distributed File System, or DFS, prefix refers to the ″scheme cellname filespace-junction″ part of the URL specification.)

Examples of prefix are:

```
"http://server"
"file://server"
"file:"
"http://server:80"
"dfs://.../cellname/fs"
```

If no prefix is found in the column data, and a default prefix is specified with DL_URL_DEFAULT_PREFIX, the default prefix is prefixed to the column value (if not NULL).

For example, if DL_URL_DEFAULT_PREFIX specifies the default prefix "http://toronto":

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://coyote/a/b/c″.
- The column input value NULL is stored as NULL.

DL_URL_REPLACE_PREFIX ″prefix″

This clause is useful when loading or importing data previously generated by the export utility, if the user wants to globally replace the host name in the data with another host name. If specified, it becomes the prefix for *all* non-NULL column values. If a column value has a prefix, this will replace it. If a column value has no prefix, the prefix specified by DL_URL_REPLACE_PREFIX is prefixed to the column value. For Distributed File System (DFS), prefix refers to the ″scheme cellname filespace-junction″ part of the URL specification.

For example, if DL_URL_REPLACE_PREFIX specifies the prefix "http://toronto":

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://toronto/a/b/c″. Note that ″toronto″ replaces ″coyote″.
- The column input value NULL is stored as NULL.

DL_URL_SUFFIX ″suffix″

If specified, it is appended to every non-NULL column value for the column. It is, in fact, appended to the ″path″ component of the data location part of the DATALINK value.

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. DATALINK exceptions are also captured in the exception table. 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 partitions on which the loading table is defined. The dump file, on the other hand, contains rows that cannot be loaded because they are invalid or have syntax errors.

FROM filename/pipename/device/cursorname

Specifies the file, pipe, device, or cursor referring to an SQL statement that contains the data being loaded. If the input source is a file, pipe, or device, it must reside on the database partition where the database resides, unless the CLIENT option is specified. If several names are specified, they will be processed in sequence. If the last item specified 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, when a new tape has been mounted).

d Device terminate. Stop using the device that generated the warning message (for example, when there are no more tapes).

t Terminate. Terminate all devices.

Notes:

1. It is recommended that the fully qualified file name be used. If the server is remote, the fully qualified file name must be used. If the database resides on the same database partition as the caller, relative paths may be used.
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.)
3. If loading data that resides on a client machine, the data must be in the form of either a fully qualified file or a named pipe.

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.

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 (*logretain* or *userexit* is enabled).
- 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 may force a rebuild, or indexes may 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.

Note: Deferred indexing is not supported for tables that have DATALINK columns.

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.

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,

```
db2 load from delfile1 of del modified by noeofchar noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, Int 4, I5, I6, DT7, I8, TM9)
```

will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

```
db2 load from delfile1 of del modified by noeofchar noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, "Int 4", I5, I6, DT7, I8, TM9)
```

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.

LOBS FROM lob-path

The path to the data files containing LOB values to be loaded. The path must end with a slash (/). If the CLIENT option is specified, the path must be fully qualified. 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. This option is ignored if `lobsinfile` is not specified within the *filetype-mod* string.

This option is not supported in conjunction with the CURSOR filetype.

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. 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 may force applications holding conflicting locks 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.

ALLOW READ ACCESS loads may force applications holding conflicting locks 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.

MESSAGES message-file

Specifies the destination for warning and error messages that occur during the load operation. If a message file is not specified, messages are written to standard output. If the complete path to the file is not specified, the load utility uses the current directory and the default drive as the destination. If the name of a file that already exists is specified, the utility appends the information.

The message file is usually populated with messages at the end of the load operation and, as such, is not suitable for monitoring the progress of the operation.

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.

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

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.

Note: 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.

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

MODIFIED BY filetype-mod

Specifies additional options.

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.

This option should not be used when DATALINK columns with the FILE LINK CONTROL attribute are present in, or being added to, the table.

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.

OF filetype

Specifies the format of the data:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format)
- IXF (integrated exchange format, PC version), exported from the same or from another DB2 table
- CURSOR (a cursor declared against a SELECT or VALUES statement).

PARTITIONED DB CONFIG

Allows you to execute a Load into a partitioned table. The

PARTITIONED DB CONFIG parameter allows you to specify partitioned database-specific configuration options. The partitioned-db-option values may be any of the following:

```
HOSTNAME x
FILE_TRANSFER_CMD x
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
```

- If Load is executed from within a non-partitioned environment, it will behave as usual. If a partitioned database configuration option is specified, this will result in SQL Error 27959, reason code 1.
- Without any further qualification, in a partitioned database environment, the MODE option will default to PARTITION_AND_LOAD except when the DB2_PARTITIONEDLOAD_DEFAULT registry variable is set to NO. In this case, the following defaults will apply: the MODE will be LOAD_ONLY, OUTPUT_DBPARTNUMS will be a list containing the single database partition to which the user is currently connected, and PART_FILE_LOCATION will be the current working path of the client, if the load input file name is not fully qualified and the client and server are on the same physical machine, or the path prefix of the load input file if that file name is fully qualified. The purpose of this registry variable is to preserve the pre-Version 8 behaviour of the Load utility in a partitioned database environment.
- In a partitioned database environment, if Load is used with the MODE option set to PARTITION_ONLY, the input file will be partitioned, resulting in the creation of a file on each output partition containing a partition map header and data for only that partition. For all file types except CURSOR, the name of the file that is created on each output partition is < filename >.< xxx >, where < filename > is the name of the input file specified in the load command, and < xxx > is the number of the partition on which the file resides. Furthermore, the location of the file on each output partition is indicated by the PART_FILE_LOCATION option, if it is specified. If this option is not specified, the location of the input file is taken to be the current working directory. If the file type is

CURSOR, the PART_FILE_LOCATION option is required and must specify a fully qualified basename. In this case, the name of the file created on each partition will be this basename appended with the appropriate partition number.

- In a partitioned database environment, if Load is used with the MODE option set to LOAD_ONLY, the files to be loaded are assumed to exist on each output partition and are assumed to contain a valid partition map header. For all file types except CURSOR, the name of the file on each partition is expected to be < filename > . < xxx >, where < filename > is the name of the input file specified in the load command, and < xxx > is the number of the partition on which the file resides. Furthermore, the location of the file on each partition is indicated by the PART_FILE_LOCATION option, if it is specified. If this option is not specified, the files will be read from the location indicated by the path prefix of the input file name, if that name is fully qualified, or the current working directory, if the input file name is not fully qualified. If the file type is CURSOR, the PART_FILE_LOCATION option is required and must specify a fully qualified basename. In this case, the name of the file on each partition is expected to be this basename appended with the appropriate partition number.
- In a partitioned database environment, if Load is used with the MODE option set to LOAD_ONLY_VERIFY_PART, the files to be loaded are assumed to exist on each output partition and are also assumed to contain no partition map header. Load will verify that the data in each file is on the proper partition. Rows that are not on the correct partition will be rejected and sent to a dumpfile, if one is specified. The name and location of the file on each output partition follows the same rules as the filename for the LOAD_ONLY mode.

  Note: The LOAD_ONLY_VERIFY_PART mode is not supported when the file type is CURSOR.
- If the CLIENT keyword of Load is specified, a remote Load will be permitted. Only the PARTITION_AND_LOAD and PARTITION_ONLY modes are supported for loads where CLIENT is specified.

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.

This option is not supported for tables with DATALINK columns.

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.

RESTARTCOUNT

Reserved.

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.

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 not supported in conjunction with the CURSOR filetype.

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.

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.

STATISTICS YES

Specifies that statistics are to be collected for the table and for any existing indexes. This option is supported only if the load operation is in REPLACE mode.

WITH DISTRIBUTION

Specifies that distribution statistics are to be collected.

AND INDEXES ALL

Specifies that both table and index statistics are to be collected.

FOR INDEXES ALL

Specifies that only index statistics are to be collected.

DETAILED

Specifies that extended index statistics are to be collected.

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:

- 4 bytes for each duplicate or rejected row containing DATALINK values
- 136 bytes for each message that the load utility generates
- 15KB 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.

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 may 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.

The load terminate option will not remove a backup pending state from table spaces.

Note: This option is not supported for tables with DATALINK columns.

USING directory

Reserved.

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 *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.

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.

If this option is not specified, and the tape device encounters an end of tape for the copy image, or the last item listed is a tape device, the user is prompted for a new tape on that device.

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
- 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 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)
```

Notes:

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 new-line character at the end of each input record, and that each record is 40 bytes long. The last 8 bytes are not used 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 (Loading LOBs from Files)

TABLE1 has 3 columns:
- COL1 CHAR 4 NOT NULL WITH DEFAULT
- LOB1 LOB
- LOB2 LOB

ASCFILE1 has 3 elements:
- ELE1 positions 01 to 04
- ELE2 positions 06 to 13
- ELE3 positions 15 to 22

The following files reside in either `/u/user1` or `/u/user1/bin`:
- ASCFILE2 has LOB data
- ASCFILE3 has LOB data
- ASCFILE4 has LOB data
- ASCFILE5 has LOB data
- ASCFILE6 has LOB data
- ASCFILE7 has LOB data

Data Records in ASCFILE1:

```
1...5....10...15...20...25...30.
REC1 ASCFILE2 ASCFILE3
REC2 ASCFILE4 ASCFILE5
REC3 ASCFILE6 ASCFILE7
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc
   lobs from /u/user1, /u/user1/bin
   modified by lobsinfile reclen=22
   method L (1 4, 6 13, 15 22)
   insert into table1
```

Notes:

1. The specification of `lobsinfile` in the MODIFIED BY parameter tells the loader that all LOB data is to be loaded from files.
2. The specification of `reclen=22` in the MODIFIED BY parameter indicates that there is no new-line character at the end of each input record, and that each record is 22 bytes long.
3. LOB data is contained in 6 files, ASCFILE2 through ASCFILE7. Each file contains the data that will be used to load a LOB column for a specific row. The relationship between LOBs and other data is specified in ASCFILE1. The first record of this file tells the loader to place REC1 in COL1 of row 1. The contents of ASCFILE2 will be used to load LOB1 of

row 1, and the contents of ASCFILE3 will be used to load LOB2 of row 1. Similarly, ASCFILE4 and ASCFILE5 will be used to load LOB1 and LOB2 of row 2, and ASCFILE6 and ASCFILE7 will be used to load the LOBs of row 3.

4. The LOBS FROM parameter contains 2 paths that will be searched for the named LOB files when those files are required by the loader.
5. To load LOBs directly from ASCFILE1 (a non-delimited ASCII file), without the `lobsinfile` modifier, the following rules must be observed:
   - The total length of any record, including LOBs, cannot exceed 32KB.
   - LOB fields in the input records must be of fixed length, and LOB data padded with blanks as necessary.
   - The `striptblanks` modifier must be specified, so that the trailing blanks used to pad LOBs can be removed as the LOBs are inserted into the database.

Example 3 (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 4 (Loading DATALINK Data)

The following command loads the table MOVIETABLE from the input file `delfile1`, which has data in the DEL format:

```
db2 load from delfile1 of del
   modified by dldel|
   insert into movietable (actorname, description, url_making_of,
   url_movie) datalink specification (dl_url_default_prefix
   "http://narang"), (dl_url_replace_prefix "http://bomdel"
   dl_url_suffix ".mpeg") for exception excptab
```

Notes:

1. The table has four columns:
   ```
   actorname            VARCHAR(n)
   description          VARCHAR(m)
   url_making_of        DATALINK (with LINKTYPE URL)
   url_movie            DATALINK (with LINKTYPE URL)
   ```
2. The DATALINK data in the input file has the vertical bar (|) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, ″http://narang″ is used.
4. Each non-NULL column value for url_movie will get ″http://bomdel″ as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ″.mpeg″ appended to the path. For example, if a column value of url_movie is ″http://server1/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″; if the value is ″/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″.
6. If any unique index or DATALINK exception occurs while loading the table, the affected records are deleted from the table and put into the exception table `excptab`.

Example 5 (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
```

Notes:

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 6 (Loading using the CURSOR filetype)

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 commands will load all the data from ABC.TABLE1 into ABC.TABLE2:

```
db2 declare mycurs cursor for select two,one,three from abc.table1
db2 load from mycurs of cursor insert into abc.table2
```

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 check pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in check pending state. Issue the SET INTEGRITY statement to take the tables out of check pending state. Load operations cannot be carried out on replicated summary 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.

DB2 Data Links Manager Considerations

For each DATALINK column, there can be one column specification within parentheses. Each column specification consists of one or more of DL_LINKTYPE, *prefix* and a DL_URL_SUFFIX specification. The *prefix* information can be either DL_URL_REPLACE_PREFIX, or the DL_URL_DEFAULT_PREFIX specification.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns as found within the insert-column list (if specified by INSERT INTO (insert-column, ...)), or within the table definition (if insert-column is not specified).

For example, if a table has columns C1, C2, C3, C4, and C5, and among them only columns C2 and C5 are of type DATALINK, and the insert-column list is (C1, C5, C3, C2), there should be two DATALINK column specifications. The first column specification will be for C5, and the second column specification will be for C2. If an insert-column list is not specified, the first column specification will be for C2, and the second column specification will be for C5.

If there are multiple DATALINK columns, and some columns do not need any particular specification, the column specification should have at least the parentheses to unambiguously identify the order of specifications. If there are no specifications for any of the columns, the entire list of empty parentheses can be dropped. Thus, in cases where the defaults are satisfactory, there need not be any DATALINK specification.

If data is being loaded into a table with a DATALINK column that is defined with FILE LINK CONTROL, perform the following steps before invoking the load utility. (If all the DATALINK columns are defined with NO LINK CONTROL, these steps are not necessary).

1. Ensure that the DB2 Data Links Manager is installed on the Data Links servers that will be referred to by the DATALINK column values. For Distributed File Systems (DFS), ensure that the DB2 Data Links Managers within the target cell are registered.
2. Ensure that the database is registered with the DB2 Data Links Manager.
3. Copy to the appropriate Data Links servers, all files that will be inserted as DATALINK values.
4. Define the prefix name (or names) to the DB2 Data Links Managers on the Data Links servers.
5. Register the Data Links servers referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file. For DFS, register the cells at the target configuration referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file.

The connection between DB2 and the Data Links server may fail while running the load utility, causing the load operation to fail. If this occurs:

1. Start the Data Links server and the DB2 Data Links Manager.
2. Invoke a load restart operation.

Links that fail during the load operation are considered to be data integrity violations, and are handled in much the same way as unique index violations. Consequently, a special exception has been defined for loading tables that have one or more DATALINK columns.

Representation of DATALINK Information in an Input File

The LINKTYPE (currently only URL is supported) is not specified as part of DATALINK information. The LINKTYPE is specified in the LOAD or the IMPORT command, and for input files of type PC/IXF, in the appropriate column descriptor records.

The syntax of DATALINK information for a URL LINKTYPE is as follows:

```
►►──┬─────────┬──┬───────────────────────┬──────────────────────────────────►◄
    └─urlname─┘  └─dl_delimiter──comment─┘
```

Note that both *urlname* and *comment* are optional. If neither is provided, the NULL value is assigned.

urlname

The URL name must conform to valid URL syntax.

Notes:

1. Currently "http", "file", "unc", and "dfs" are permitted as a schema name.
2. The prefix (schema, host, and port) of the URL name is optional. For DFS, the prefix refers to the schema cellname filespace-junction portion. If a prefix is not present, it is taken from the DL_URL_DEFAULT_PREFIX or the DL_URL_REPLACE_PREFIX specification of the load or the import utility. If none of these is specified, the prefix defaults to "file://localhost". Thus, in the case of local files, the file name with full path name can be entered as the URL name, without the need for a DATALINK column specification within the LOAD or the IMPORT command.
3. Prefixes, even if present in URL names, are overridden by a different prefix name on the DL_URL_REPLACE_PREFIX specification during a load or import operation.
4. The "path" (after appending DL_URL_SUFFIX, if specified) is the full path name of the remote file in the remote server. Relative path names are not allowed. The http server default path-prefix is not taken into account.

dl_delimiter

For the delimited ASCII (DEL) file format, a character specified via the `dldel` modifier, or defaulted to on the LOAD or the IMPORT command. For the non-delimited ASCII (ASC) file format, this should correspond to the character sequence **\;** (a backslash followed by a semicolon). Whitespace characters (blanks, tabs, and so on) are permitted before and after the value specified for this parameter.

comment

The comment portion of a DATALINK value. If specified for the delimited ASCII (DEL) file format, the *comment* text must be enclosed by the character string delimiter, which is double quotation marks (″) by default. This character string delimiter can be overridden by the MODIFIED BY *filetype-mod* specification of the LOAD or the IMPORT command.

If no comment is specified, the comment defaults to a string of length zero.

Following are DATALINK data examples for the delimited ASCII (DEL) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg; "Intro Movie"`

  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang; "InderPal's Home Page"`

  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal's Home Page″

Following are DATALINK data examples for the non-delimited ASCII (ASC) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg\;Intro Movie`

  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang\; InderPal's Home Page`

  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal's Home Page″

Following are DATALINK data examples in which the load or import specification for the column is assumed to be DL_URL_REPLACE_PREFIX (″http://qso″):

- `http://www.almaden.ibm.com/mrep/intro.mpeg`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /mrep/intro.mpeg
  - comment = NULL string
- `/u/me/myfile.ps`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /u/me/myfile.ps
  - comment = NULL string

Table 8. Valid File Type Modifiers (LOAD)

Modifier	**Description**
All File Formats	
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.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
fastparse	Reduced syntax checking is done on user-supplied column values, and performance is enhanced. Tables loaded under this option are guaranteed to be architecturally correct, and the utility is guaranteed to perform sufficient data checking to prevent a segmentation violation or trap. Data that is in correct form will be loaded correctly. For example, if a value of `123qwr4` were to be encountered as a field entry for an integer column in an ASC file, the load utility would ordinarily flag a syntax error, since the value does not represent a valid number. With `fastparse`, a syntax error is not detected, and an arbitrary number is loaded into the integer field. Care must be taken to use this modifier with clean data only. Performance improvements using this option with ASCII data can be quite substantial, but `fastparse` does not significantly enhance performance with PC/IXF data, since IXF is a binary format, and `fastparse` affects parsing and conversion from ASCII to internal forms. This option is not supported in conjunction with the CURSOR filetype.
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), and will therefore load NULLs into the column. 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 8. Valid File Type Modifiers (LOAD) (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).
	Note: When this modifier is used, the table will be placed in CHECK PENDING state. To take the table out of CHECK PENDING state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > GENERATED COLUMN IMMEDIATED UNCH`
	To take the table out of CHECK 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.`
	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.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	**Description**
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. **Note:** 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 INDEX ING MODE INCREMENTAL ignores this option. The first entry in a page is added without restriction; subsequent entries are added the percent free space threshold can be maintained. 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 registry variable DB2 INDEX FREE takes precedence over indexfreespace. The indexfreespace option affects index leaf pages only.

Table 8. Valid File Type Modifiers (LOAD) (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. 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). The AutoLoader utility writes a header to each file contributing data to a table in a multiple-partition database partition group. If the default MPP load (mode PARTITION_AND_LOAD) is used against a table residing in a single node nodegroup 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.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
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 3á000 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. **Note:** 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.
subtableconvert	Valid only when loading into a single sub-table. Typical usage is to export data from a regular table, and then to invoke a load operation (using this modifier) to convert the data into a single sub-table.
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. **Note:** If two loads are done with this option specified, the second load will not 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: ",," is specified for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	

Table 8. Valid File Type Modifiers (LOAD) (continued)

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 may 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.

dateformat="*x*"	*x* is the format of the date in the source file.[a] 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 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

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 32KB 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`
	Notes:
	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. This modifier does not support file names with multiple file extensions. For example,
	`dumpfile = /home/svtdbm6/DUMP.FILE`
	is acceptable to the load utility, but
	`dumpfile = /home/svtdbm6/DUMP.LOAD.FILE`
	is not.

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`.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

timeformat=*"x"*	*x* is the format of the time in the source file.[a] 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 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[a] 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 1 - 12; mutually exclusive with M, month)`
	`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)`
	`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 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 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 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.
ASC (Non-delimited ASCII) File Format	
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 `noeofchar` option is assumed. 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. **Note:** 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`.[b] 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 8. Valid File Type Modifiers (LOAD) (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. The `noeofchar` option is assumed.
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	Note: 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.

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. **Note:** This option replaces the obsolete `t` option, which is supported for back-level 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. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

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. The NOEOFCHAR option is assumed.
	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`.

DEL (Delimited ASCII) File Format	

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.[bc] If you wish 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.[bc]
datesiso	Date format. Causes all date data values to be loaded in ISO format.
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.[bc]

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	**Description**

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>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [bcd] **Note:** *x* must not be the same character specified as the row, column, or character string 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 tailing 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`

nodoubledel	Suppresses recognition of double character delimiters.
IXF File Format	

Table 8. Valid File Type Modifiers (LOAD) (continued)

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.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

Notes:

1. 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.
2. [a] 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)
   ```

   Note: Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
3. [b] The character 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. [c] Delimiter Restrictions lists restrictions that apply to the characters that can be used as delimiter overrides.
5. [d] Even if the DATALINK delimiter character is a valid character within the URL syntax, it will lose its special meaning within the scope of the load operation.

Related reference:

- "QUIESCE TABLESPACES FOR TABLE" in the *Command Reference*
- "Partitioned Database Load Configuration Options" on page 262

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:

```
>>—LOAD QUERY—TABLE—table-name—┬─────────────────────────┬─┬─────────────┬──────>
                               └─TO—local-message-file─┘ ├─NOSUMMARY───┤
                                                         └─SUMMARYONLY─┘

>─┬───────────┬─────────────────────────────────────────────────────────────><
  └─SHOWDELTA─┘
```

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 wants to check the status of the load operation. The user can specify:

```
   db2 connect to <database>
   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 may be used on tables that are not currently being loaded. The table states described by LOAD QUERY are as follows:

Normal
No table states affecting the table.

Check Pending
The table has constraints and the constraints have yet to be verified. Use the SET INTEGRITY command to take the table out of Check Pending. The load utility puts a table into the Check Pending state when it begins a load on a table with constraints.

Load in Progress
There is a load actively in progress on this table.

Load Pending
A load has been active on this table but has been aborted before the load could commit. Issue a load terminate, a load restart or a load replace to bring the table out of the Load Pending state.

Read Access Only
The table data is available for read access queries. Loads using the ALLOW READ ACCESS option put the table into Read Access Only state.

Unavailable
The table is unavailable. The table may only be dropped or it may be restored from a backup. Rollforward through a non-recoverable load will put a table into the unavailable state.

Not Load Restartable
The table is in a partially loaded state that will not allow a load restart. The table will also be in the Load Pending state. Issue a load terminate or a load replace to bring the table out of the Not Load Restartable state. The 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 aborted load operation but do not roll forward to the end of the load terminate or load restart operation.

Unknown
Load query is unable to determine a table state.

Related reference:

- "LOAD" on page 133

db2Load - Load

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. The load utility does not support loading data at the hierarchy level.

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.

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

C API syntax:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
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;
} db2LoadStruct;

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;
} 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;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
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 db2LoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2gPartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   db2Uint16 iFileTypeLen;
   db2Uint16 iLocalMsgFileLen;
   db2Uint16 iTempFilesPathLen;
} db2gLoadStruct;

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;
} db2LoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64                           oRowsRead;
   db2Uint64                           oRowsSkipped;
   db2Uint64                           oRowsLoaded;
   db2Uint64                           oRowsRejected;
   db2Uint64                           oRowsDeleted;
   db2Uint64                           oRowsCommitted;
} db2LoadOut;

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;
   SQL_PDB_NODE_TYPE                   *piRunStatDBPartNum;
   db2Uint16                           iHostnameLen;
   db2Uint16                           iFileTransferLen;
   db2Uint16                           iPartFileLocLen;
   db2Uint16                           iMapFileInputLen;
   db2Uint16                           iMapFileOutputLen;
   db2Uint16                           iDistfileLen;
} db2gPartLoadIn;

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;
/* ... */
```

API parameters:

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

pParamStruct
Input. A pointer to the *db2LoadStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

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`) 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.

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`) 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 *pFileType* parameter is set to `SQL_ASC`, the *dcolmeth* field of this structure must either be set to `SQL_METH_L` or be set to `SQL_METH_D` and specifies a file name with POSITIONSFILE *pFileTypeMod* modifier which contains starting and ending pairs and null indicator positions. 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
Input. Pointer to an *sqlchar* structure, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE|RESTART|TERMINATE
INTO tbname [(column_list)]
[DATALINK SPECIFICATION datalink-spec]
[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.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the LOAD command.

FOR EXCEPTION *e_tbname*
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. DATALINK exceptions are also captured in 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 Integrated 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 of type SQLU_SQL_STMT, and refers to an actual SQL query and not a cursor declared against one.

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.

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 field (defined in `sqlutil`) 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 *pDataDescriptor* 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
Input. 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`) 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.

iFileTypeLen

Input. Specifies the length in bytes of *iFileType*.

iLocalMsgFileLen

Input. Specifies the length in bytes of *iLocalMsgFileName*.

iTempFilesPathLen

Input. Specifies the length in bytes of *iTempFilesPath*.

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 tablespace *iUseTablespaceName* and copied over to the original tablespace 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 tablespace as the index object.

If the shadow copy is created in the same tablespace 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 tablespace 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 *savecnt* 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 *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 spawn 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 spawn 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`) 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 (i.e., 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
Input. Specifies to put the table into check pending state. If SQLU_CP_IMMEDIATE is specified, check pending state will be immediately cascaded to all dependent and descendent tables. If SQLU_CP_DEFERRED is specified, the cascade of check pending state to dependent tables will be deferred until the target table is checked for integrity violations. SQLU_CP_DEFERRED is the default if the option is not specified.

iRestartphase
Input. The restart phase.

iStatsOpt
Input. Granularity of statistics to collect. Valid values are:

SQLU_STATS_NONE
No statistics to be gathered.

SQL_STATS_INDEX
Basic index statistics without table statistics.

SQL_STATS_TABLE
Basic table statistics without index statistics.

SQL_STATS_BOTH
Basic table and index statistics.

SQL_STATS_EXTTABLE_ONLY
Table and distribution statistics.

SQL_STATS_EXTTABLE_INDEX
Table and distribution statistics and basic index statistics.

SQL_STATS_EXTINDEX_ONLY
Extended statistics for indexes only.

SQL_STATS_EXTINDEX_TABLE
Extended statistics for indexes and basic table statistics.

SQL_STATS_ALL
Extended index statistics with table and distribution statistics.

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.

piHostname
Input. The hostname for the *iFileTransferCmd* parameter. If NULL, the hostname will default to "nohost".

piFileTransferCmd
Input. File transfer command parameter. If not required, it must be set to NULL. See the Data Movement Guide for a full description of this parameter.

piPartFileLocation
Input. In PARTITION_ONLY, LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the fully qualified location of the partitioned files. This location must exist on each partition specified by the *piOutputNodes* option.

For file types other than SQL_CURSOR, the following rules apply: In PARTITION_ONLY mode, if the value of this parameter is NULL, the current directory will be used as the location where load will create the partitioned files. For LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, if this parameter is NULL, the location of the input files will be taken as the path prefix of the input file name specified in the piSourceList parameter, as long as that name is fully-qualified; otherwise, the current directory will be used as the location of the input files.

For the SQL_CURSOR file type, this parameter cannot by 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 partition for PARTITION_ONLY mode, or the location of the files to be read from each 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.

piOutputNodes
Input. The list of Load output 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. Refer to the Load command in the Data Movement Guide and Reference for a description of how the default is determined.

piMode
Input. Specifies the load mode for partitioned databases. Valid values (defined in db2ApiDf) are:

PARTITION_AND_LOAD
Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

PARTITION_ONLY
Data is partitioned (perhaps in parallel) and the output is written to files in a specified location on each loading partition. See the description of the *piPartFileLocation* parameter for a description of how to specify the location of the output files.

LOAD_ONLY
Data is assumed to be already partitioned; the partition process is skipped, and the data is loaded simultaneously on the corresponding database partitions. See the description of the *piPartFileLocation* parameter for a description of how to specify the location of the input files.

LOAD_ONLY_VERIFY_PART
Data is assumed to be already partitioned, but the data file does not contain a partition header. The partitioning 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 partition. Rows containing partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If partition violations exist on a particular

loading partition, a single warning will be written to the load message file for that partition. See the description of the *piPartFileLocation* parameter for a description of how to specify the location of the input files.

ANALYZE

An optimal partitioning 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 partitions. Valid values (defined in `db2ApiDf`) are:

DB2LOAD_SETUP_ERRS_ONLY

In this mode, errors that occur on a partition during setup, such as problems accessing a partition or problems accessing a table space or table on a partition, will cause the load operation to stop on the failing partitions but to continue on the remaining partitions. Errors that occur on a partition while data is being loaded will cause the entire operation to fail and rollback to the last point of consistency on each partition.

DB2LOAD_LOAD_ERRS_ONLY

In this mode, errors that occur on a partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining partitions until a failure occurs or until all the data is loaded. On the 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 partitions the transaction will be aborted. Data on all of the partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the partitions where the load operation completed and resume the load operation on partitions that experienced an error.

Note: This mode cannot be used when *iAccessLevel* is set to SQLU_ALLOW_READ_ACCESS or a copy target is specified.

DB2LOAD_SETUP_AND_LOAD_ERRS

In this mode, partition-level errors during setup or loading data cause processing to stop only on the affected partitions.

As with the DB2LOAD_LOAD_ERRS_ONLY mode, when partition errors do occur while data is being loaded, the data on all partitions will remain invisible until a load restart operation is performed.

Note: This mode cannot be used when *iAccessLevel* is set to SQLU_ALLOW_READ_ACCESS or a copy target is 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 or a copy target is 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. Partition 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. Partition 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 partition distribution file. If a NULL is specified, the value defaults to ″DISTFILE″.

piOmitHeader
Input. Indicates that a partition map header should not be included in the 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 partition list.

iHostnameLen
Input. The length in bytes of *piHostname*.

iFileTransferLen
Input. The length in bytes of *piFileTransferCmd*.

iPartFileLocLen
Input. The length in bytes of *piPartFileLocation*.

iMapFileInputLen
Input. The length in bytes of *piMapFileInput*.

iMapFileOutputLen
Input. The length in bytes of *piMapFileOutput*.

iDistfileLen
Input. The length in bytes of *piDistfile*.

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.

iPortMin
Input. Lower port number.

iPortMax
Input. Higher port number.

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-partitioing 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 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 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 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*.

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 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 CHECK 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 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 partitions.

DB2LOADQUERY_LOADPENDING
Indicates that the load job aborted during processing but left the table on the partition in the LOAD PENDING state, indicating that the load job on that partition must be either terminated or restarted.

oNodeNum
Output. The number of the partition on which the agent executed.

oAgentType
Output. The agent type. Valid values (defined in `db2ApiDf`) are :

DB2LOAD_LOAD_AGENT

DB2LOAD_PARTITIONING_AGENT

DB2LOAD_PRE_PARTITIONING_AGENT

DB2LOAD_FILE_TRANSFER_AGENT

DB2LOAD_LOAD_TO_FILE_AGENT

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 check pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in check pending state. Issue the SET INTEGRITY statement to take the tables out of check 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.

DB2 Data Links Manager Considerations

For each DATALINK column, there can be one column specification within parentheses. Each column specification consists of one or more of DL_LINKTYPE, *prefix* and a DL_URL_SUFFIX specification. The *prefix* information can be either DL_URL_REPLACE_PREFIX, or the DL_URL_DEFAULT_PREFIX specification.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns as found within the insert-column list (if specified by INSERT INTO (insert-column, ...)), or within the table definition (if insert-column is not specified).

For example, if a table has columns C1, C2, C3, C4, and C5, and among them only columns C2 and C5 are of type DATALINK, and the insert-column list is (C1, C5, C3, C2), there should be two DATALINK column specifications. The first column specification will be for C5, and the second column specification will be for C2. If an insert-column list is not specified, the first column specification will be for C2, and the second column specification will be for C5.

If there are multiple DATALINK columns, and some columns do not need any particular specification, the column specification should have at least the parentheses to unambiguously identify the order of specifications. If there are no specifications for any of the columns, the entire list of empty parentheses can be dropped. Thus, in cases where the defaults are satisfactory, there need not be any DATALINK specification.

If data is being loaded into a table with a DATALINK column that is defined with FILE LINK CONTROL, perform the following steps before invoking the load utility. (If all the DATALINK columns are defined with NO LINK CONTROL, these steps are not necessary).

1. Ensure that the DB2 Data Links Manager is installed on the Data Links servers that will be referred to by the DATALINK column values. For Distributed File Systems (DFS), ensure that the DB2 Data Links Managers within the target cell are registered.

2. Ensure that the database is registered with the DB2 Data Links Manager.
3. Copy to the appropriate Data Links servers, all files that will be inserted as DATALINK values.
4. Define the prefix name (or names) to the DB2 Data Links Managers on the Data Links servers.
5. Register the Data Links servers referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file. For DFS, register the cells at the target configuration referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file.

The connection between DB2 and the Data Links server may fail while running the load utility, causing the load operation to fail. If this occurs:

1. Start the Data Links server and the DB2 Data Links Manager.
2. Invoke a load restart operation.

Links that fail during the load operation are considered to be data integrity violations, and are handled in much the same way as unique index violations. Consequently, a special exception has been defined for loading tables that have one or more DATALINK columns.

Representation of DATALINK Information in an Input File

The LINKTYPE (currently only URL is supported) is not specified as part of DATALINK information. The LINKTYPE is specified in the LOAD or the IMPORT command, and for input files of type PC/IXF, in the appropriate column descriptor records.

The syntax of DATALINK information for a URL LINKTYPE is as follows:

```
>>-+---------+--+-----------------------+--------------------------------><
   '-urlname-'  '-dl_delimiter--comment-'
```

Note that both *urlname* and *comment* are optional. If neither is provided, the NULL value is assigned.

urlname

The URL name must conform to valid URL syntax.

Notes:

1. Currently "http", "file", "unc", and "dfs" are permitted as a schema name.
2. The prefix (schema, host, and port) of the URL name is optional. For DFS, the prefix refers to the schema cellname filespace-junction portion. If a prefix is not present, it is taken from the

DL_URL_DEFAULT_PREFIX or the DL_URL_REPLACE_PREFIX specification of the load or the import utility. If none of these is specified, the prefix defaults to ″file://localhost″. Thus, in the case of local files, the file name with full path name can be entered as the URL name, without the need for a DATALINK column specification within the LOAD or the IMPORT command.

3. Prefixes, even if present in URL names, are overridden by a different prefix name on the DL_URL_REPLACE_PREFIX specification during a load or import operation.
4. The ″path″ (after appending DL_URL_SUFFIX, if specified) is the full path name of the remote file in the remote server. Relative path names are not allowed. The http server default path-prefix is not taken into account.

dl_delimiter

For the delimited ASCII (DEL) file format, a character specified via the `dldel` modifier, or defaulted to on the LOAD or the IMPORT command. For the non-delimited ASCII (ASC) file format, this should correspond to the character sequence **\;** (a backslash followed by a semicolon). Whitespace characters (blanks, tabs, and so on) are permitted before and after the value specified for this parameter.

comment

The comment portion of a DATALINK value. If specified for the delimited ASCII (DEL) file format, the *comment* text must be enclosed by the character string delimiter, which is double quotation marks (″) by default. This character string delimiter can be overridden by the MODIFIED BY *filetype-mod* specification of the LOAD or the IMPORT command.

If no comment is specified, the comment defaults to a string of length zero.

Following are DATALINK data examples for the delimited ASCII (DEL) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg; "Intro Movie"`

  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang; "InderPal's Home Page"`

  This is stored with the following parts:
  - scheme = file

- server = narang
- path = /u/narang
- comment = ″InderPal′s Home Page″

Following are DATALINK data examples for the non-delimited ASCII (ASC) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg\;Intro Movie`

  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang\; InderPal's Home Page`

  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal′s Home Page″

Following are DATALINK data examples in which the load or import specification for the column is assumed to be DL_URL_REPLACE_PREFIX (″http://qso″):

- `http://www.almaden.ibm.com/mrep/intro.mpeg`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /mrep/intro.mpeg
  - comment = NULL string
- `/u/me/myfile.ps`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /u/me/myfile.ps
  - comment = NULL string

Table 9. Valid File Type Modifiers (LOAD)

Modifier	**Description**
All File Formats	

Table 9. Valid File Type Modifiers (LOAD) (continued)

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.
fastparse	Reduced syntax checking is done on user-supplied column values, and performance is enhanced. Tables loaded under this option are guaranteed to be architecturally correct, and the utility is guaranteed to perform sufficient data checking to prevent a segmentation violation or trap. Data that is in correct form will be loaded correctly. For example, if a value of `123qwr4` were to be encountered as a field entry for an integer column in an ASC file, the load utility would ordinarily flag a syntax error, since the value does not represent a valid number. With `fastparse`, a syntax error is not detected, and an arbitrary number is loaded into the integer field. Care must be taken to use this modifier with clean data only. Performance improvements using this option with ASCII data can be quite substantial, but `fastparse` does not significantly enhance performance with PC/IXF data, since IXF is a binary format, and `fastparse` affects parsing and conversion from ASCII to internal forms. This option is not supported in conjunction with the CURSOR filetype.
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 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

generatedoverride	This modifier instructs the load utility to accept explicit, non-NULL 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).
	Note: When this modifier is used, the table will be placed in CHECK PENDING state. To take the table out of CHECK PENDING state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR <table-name> GENERATED  COLUMN IMMEDIATED UNCH`
	To take the table out of CHECK 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`

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.

Table 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
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. **Note:** 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 loading the index. The first entry in a page is added without restriction; subsequent entries are added if the percent free space threshold can be maintained. 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 registry variable DB2 INDEX FREE takes precedence over indexfreespace. The indexfreespace option affects index leaf pages only.

Table 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB values. 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. 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). The AutoLoader utility writes a header to each file contributing data to a table in a multiple-partition database partition group. The header includes the database partition number, the partitioning map, and the partitioning key specification. The load utility requires this information to verify that the data is being loaded at the correct database partition. When loading files into a table that exists on a single-partition database partition group, the headers do not exist, and this option causes the load utility to skip the header verification code.
norowwarnings	Suppresses all warnings about rejected rows.

Table 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
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 3000 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. **Note:** 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.
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. **Note:** If two loads are done with this option specified, the second load will not 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: ",," is specified for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	

Table 9. Valid File Type Modifiers (LOAD) (continued)

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 may 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.

dateformat=″*x*″	*x* is the format of the date in the source file.[a] 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 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

dumpfile = *x*	*x* is the fully qualified (according to the server node) name of an exception file to which rejected rows are written. A maximum of 32KB 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`
	Notes:
	1. In a partitioned database environment, the path should be local to the loading node, 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. This modifier does not support file names with multiple file extensions. For example,
	`dumpfile = /home/svtdbm6/DUMP.FILE`
	is acceptable to the load utility, but
	`dumpfile = /home/svtdbm6/DUMP.LOAD.FILE`
	is not.

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`.

Table 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

timeformat=*"x"*	*x* is the format of the time in the source file.[a] 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 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[a] 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 1 - 12; mutually exclusive with M, month)`
	`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)`
	`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 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 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 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.
ASC (Non-delimited ASCII) File Format	
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 `noeofchar` option is assumed. 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. **Note:** 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`.[b] 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 9. Valid File Type Modifiers (LOAD) (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. The `noeofchar` option is assumed.
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	Note: 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 the Windows operating system, the byte order must not be reversed.

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. **Note:** This option replaces the obsolete `t` option, which is supported for back-level 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. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.

Table 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

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. The NOEOFCHAR option is assumed.
	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`.

DEL (Delimited ASCII) File Format	

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.[bc]
	The single quotation mark (') can also be specified as a character string delimiter as follows:
	`modified by chardel''`
	If you wish to explicitly specify the double quotation mark(″) as the character string delimiter, you should specify it 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.[bc]
datesiso	Date format. Causes all date data values to be loaded in ISO format.
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.[bc]

Table 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

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>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [bcd] **Note:** *x* must not be the same character specified as the row, column, or character string 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 tailing 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`

nodoubledel	Suppresses recognition of double character delimiters. For more information, see the delimiter restrictions in the import API.
IXF File Format	

Table 9. Valid File Type Modifiers (LOAD) (continued)

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.

Table 9. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

Notes:

1. 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.
2. [a] 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)
   ```

   Note: Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
3. [b] The character 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. [c] The delimeter restrictions in the Import API lists restrictions that apply to the characters that can be used as delimiter overrides.
5. [d] Even if the DATALINK delimiter character is a valid character within the URL syntax, it will lose its special meaning within the scope of the load operation.

Related reference:

- "sqluvqdp - Quiesce Table Spaces for Table" in the *Administrative API Reference*
- "db2LoadQuery - Load Query" on page 223
- "SQLU-MEDIA-LIST" in the *Administrative API Reference*
- "sqluexpr - Export" on page 17
- "sqluimpr - Import" on page 67
- "db2DatabaseQuiesce - Database Quiesce" in the *Administrative API Reference*
- "db2InstanceQuiesce - Instance Quiesce" in the *Administrative API Reference*

Related samples:

- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbload.sqc -- How to load into a partitioned database (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2LoadQuery - Load Query

Checks the status of a load operation during processing.

Authorization:

None

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LoadQuery */
/* ... */
SQL_API_RC SQL_API_FN
db2LoadQuery (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  char *piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct *poOutputStruct;
  char *piLocalMessageFile;
} db2LoadQueryStruct;

typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
  db2Uint32 oTableState;
} db2LoadQueryOutputStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gLoadQuery */
/* ... */
SQL_API_RC SQL_API_FN
db2gLoadQuery (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  db2Uint32 iStringLen;
  char *piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct *poOutputStruct;
  db2Uint32 iLocalMessageFileLen;
  char *piLocalMessageFile
} db2gLoadQueryStruct;

typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
  db2Uint32 oTableState;
} db2LoadQueryOutputStruct;
/* ... */
```

API parameters:

versionNumber

Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct

Input. A pointer to the *db2LoadQueryStruct* structure.

pSqlca

Output. A pointer to the *sqlca* structure.

iStringType

Input. Specifies a type for *piString*. Valid values (defined in `db2ApiDf.h`) are:

DB2LOADQUERY_TABLENAME
Specifies a table name for use by the db2LoadQuery API.

iStringLen
Input. Specifies the length in bytes of *piString*.

piString
Input. Specifies a temporary files path name or a table name, depending on the value of *iStringType*.

iShowLoadMessages
Input. Specifies the level of messages that are to be returned by the load utility. Valid values (defined in `db2ApiDf.h`) are:

DB2LOADQUERY_SHOW_ALL_MSGS
Return all load messages.

DB2LOADQUERY_SHOW_NO_MSGS
Return no load messages.

DB2LOADQUERY_SHOW_NEW_MSGS
Return only messages that have been generated since the last call to this API.

poOutputStruct
Output. A pointer to the *db2LoadQueryOutputStruct* structure, which contains load summary information. Set to NULL if a summary is not required.

iLocalMessageFileLen
Input. Specifies the length in bytes of *piLocalMessageFile*.

piLocalMessageFile
Input. Specifies the name of a local file to be used for output messages.

oRowsRead
Output. Number of records read so far by the load utility.

oRowsSkipped
Output. Number of records skipped before the load operation began.

oRowsCommitted
Output. Number of rows committed to the target table so far.

oRowsLoaded
Output. Number of rows loaded into the target table so far.

oRowsRejected
Output. Number of rows rejected from the target table so far.

oRowsDeleted
Output. Number of rows deleted from the target table so far (during the delete phase).

oCurrentIndex
Output. Index currently being built (during the build phase).

oCurrentMPPNode
Output. Indicates which database partition server is being queried (for partitioned database environment mode only).

oLoadRestarted
Output. A flag whose value is `TRUE` if the load operation being queried is a load restart operation.

oWhichPhase
Output. Indicates the current phase of the load operation being queried. Valid values (defined in `db2ApiDf.h`) are:

DB2LOADQUERY_LOAD_PHASE
Load phase.

DB2LOADQUERY_BUILD_PHASE
Build phase.

DB2LOADQUERY_DELETE_PHASE
Delete phase.

oNumTotalIndexes
Output. Total number of indexes to be built (during the build phase).

oWarningCount
Output. Total number of warnings returned so far.

oTableState
Output. The table states. Valid values (as definde in `db2ApiDf`) are:

DB2LOADQUERY_NORMAL
No table states affect the table.

DB2LOADQUERY_CHECK_PENDING
The table has constraints and the constraints have yet to be verified. Use the SET INTEGRITY command to take the table out of the DB2LOADQUERY_CHECK_PENDING state. The load utility puts a table into the DB2LOADQUERY_CHECK_PENDING state when it begins a load on a table with constraints.

DB2LOADQUERY_LOAD_IN_PROGRESS
There is a load actively in progree on this table.

DB2LOADQUERY_LOAD_PENDING
A load has been active on this table but has been aborted

before the load could commit. Issue a load terminate, a load restart, or a load replace to bring the table out of the DB2LOADQUERY_LOAD_PENDING state.

DB2LOADQUERY_READ_ACCESS
The table data is available for read access queries. Loads using the DB2LOADQUERY_READ_ACCESS option put the table into Read Access Only state.

DB2LOADQUERY_NOTAVAILABLE
The table is unavailable. The table may only be dropped or it may be restored from a backup. Rollforward through a non-recoverable load will put a table into the unavailable state.

DB2LOADQUERY_NO_LOAD_RESTART
The table is in a partially loaded state that will not allow a load restart. The table will also be in the Load Pending state. Issue a load terminate or a load replace to bring the talbe out of the Not Load Restartable state. The table can be placed in the DB2LOADQUERY_NO_LOAD_RESTART state during a rollforward operation. This can occur if you rollforward to a point in time that is prior to the end of a load operation, or if you roll forward through an aborted load operation but do not roll forward to the end of the load terminate or load restart operation.

Usage notes:

This API reads the status of the load operation on the table specified by *piString*, and writes the status to the file specified by *pLocalMsgFileName*.

Related reference:

- “SQLCA” in the *Administrative API Reference*
- “LOAD QUERY” on page 178

Related samples:

- “loadqry.sqb -- Query the current status of a load (MF COBOL)”
- “tbmove.sqc -- How to move table data (C)”
- “tbmove.sqC -- How to move table data (C++)”

Load Exception Table

The exception table is a user-created table that reflects the definition of the table being loaded, and includes some additional columns. It is specified by the FOR EXCEPTION clause on the LOAD command. An exception table may 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. The exception table is used to store copies of rows that violate unique index rules; the utility will not check for constraints or foreign key violations other than violations of uniqueness. DATALINK exceptions are also captured in the exception table.

A load exception table can be assigned to the table space where the table being loaded resides, or to another table space. In either case, the load exception table should be assigned to the same database partition group and have the same partitioning key as the table being loaded.

A unique key is a key for which no two values are equal. The mechanism used to enforce this constraint is called a unique index. A primary key is a special case of a unique key. A table cannot have more than one primary key.

Note: Any rows rejected because of invalid data before the building of an index are not inserted into the exception table.

Rows are appended to existing information in the exception table; this can include invalid rows from previous load operations. If you want only the invalid rows from the current load operation, you must remove the existing rows before invoking the utility.

The exception table used with the load utility is identical to the exception tables used by the SET INTEGRITY statement.

An exception table should be used when loading data which has a unique index and the possibility of duplicate records. If an exception table is not specified, and duplicate records are found, the load operation continues, and only a warning message is issued about the deleted duplicate records. The records themselves are not logged.

After the load operation completes, information in the exception table can be used to correct data that is in error. The corrected data can then be inserted into the table.

Related reference:

- "Exception tables" in the *SQL Reference, Volume 1*

Load Dump File

Specifying the *dumpfile* 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, the name is given an extension that identifies the partition number where the exceptions were generated. For example:

```
dumpfile = "/u/usrname/dumpit"
```

On partition zero, this will generate a file named `/u/usrname/dumpit.000`. On partition five, it will generate a file named `/u/usrname/dumpit.005`, and so on.

If it is less than 32 768 bytes in length, the record is copied to the dump file in its entirety; if it is longer, a record fragment (including the final bytes of the record) is written to the file.

Related reference:

- "LOAD" on page 133

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. The 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 will be 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 will cause the load operation to malfunction, and will place 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 will be 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 will create depending on the size of the input data:

- Two log records will be created for every table space extent allocated or deleted by the utility in a DMS table space.
- One log record will be created for every chunk of identity values consumed.
- Log records will be created for every data row or index key deleted during the delete phase of a load operation.
- Log records will be 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.

Table Locking, Table States and Table Space States

In most cases, the load utility uses table level locking to restrict access to tables. The load utility does not quiesce the table spaces involved in the load operation, and uses table space states only for load operations with the COPY NO option specified. The level of locking depends on whether or not the load operation allows read access. A load operation in ALLOW NO ACCESS mode will use an exclusive lock (Z-lock) on the table for the duration of the load. An load operation in ALLOW READ ACCESS mode acquires and maintains a share lock (S-lock) for the duration of the load operation, and upgrades the lock to an exclusive lock (Z-lock) when data is being committed.

Before a load operation in ALLOW READ ACCESS mode begins, the load utility will wait for all applications that began before the load operation to release locks on the target table. Since locks are not persistent, they are supplemented by table states that will remain even if a load operation is aborted. These states can be checked by using the LOAD QUERY command. By using the LOCK WITH FORCE option, the load utility will force applications holding conflicting locks off the table that it is trying to load into.

Locking Behavior For Load Operations in ALLOW READ ACCESS Mode

At the beginning of a load operation, the load utility acquires a share lock (S-lock) on the table. It holds this lock until the data is being committed. The share lock allows applications with compatible locks to access the table during the load operation. For example, applications that use read only queries will be able to access the table, while applications that try to insert data into the table will be denied. When the load utility acquires the share lock on the table, it will wait 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. Since the load utility upgrades the share lock to an exclusive (Z-lock) when the data is being committed, there may be some delay in commit time while the load utility waits for applications with conflicting locks to finish.

Note: The load operation will not timeout while it waits for the applications to release their locks on the table.

LOCK WITH FORCE Option

The LOCK WITH FORCE option can be used to force off applications holding conflicting locks on a table so that the load operation can proceed. If an application is forced off the system by the load utility, it will lose its database connection and an error will be returned (SQL1224N).

For a load operation in ALLOW NO ACCESS mode, all applications holding table locks that exist at the start of the load operation will be forced.

For a load operation in ALLOW READ ACCESS mode applications holding the following locks will be forced:

- Table locks that conflict with a table share lock (e.g., import or insert).
- All table locks that exist at the commit phase of the load operation.

When the COPY NO option is specified for a load operation on a recoverable database, all objects in the target table space will be locked in share mode before the table space is placed in backup pending state. This will occur regardless of the access mode. If the LOCK WITH FORCE option is specified, all applications holding locks on objects in the table space that conflict with a share lock will be forced off.

Table States

In addition to locks, the load utility uses table states to control access to tables. A table state can be checked by using the LOAD QUERY command. The following states returned by the LOAD QUERY command are as follows:

Normal
No table states affect the table.

Check Pending
The table has constraints which have not yet been verified. Use the SET INTEGRITY statement to take the table out of check pending state. The load utility places a table in the check pending state when it begins a load operation on a table with constraints.

Load in Progress
There is a load operation in progress on this table.

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
The table data is available for read access queries. Load operations using the ALLOW READ ACCESS option place the table in read access only state.

Unavailable
The table is unavailable. The table may 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 determine the table state.

A table can be in several states at the same time. For example, if data is loaded into a table with constraints and the ALLOW READ ACCESS option is specified, table state would be:

```
Tablestate:
  Check Pending
  Load in Progress
  Read Access Only
```

After the load operation but before issuing the SET INTEGRITY statement, the table state would be:

```
Tablestate:
  Check Pending
  Read Access Only
```

After the SET INTEGRITY statement has been issued the table state would be:

```
Tablestate:
  Normal
```

Table Space States when COPY NO is Specified

If a load operation with the COPY NO option is executed in a recoverable database, the table spaces associated with the load operation are placed in the backup table space state and the load in progress table space state. This takes place at the beginning of the load operation. The load operation may be delayed at this point while locks are acquired on the tables within the table space.

When a table space is in backup pending state, it is still available for read access. The table space can only be taken out of backup pending state by taking a backup of the table space. Even if the load operation is aborted, the table space will remain in backup pending state because the table space state is changed at the beginning of the load operation, and cannot be rolled back if it fails. The load in progress table space state prevents online backups of a load operation with the COPY NO option specified while data is being loaded. The load in progress state is removed when the load operation is completed or aborts.

During a rollforward operation through a LOAD command with the COPY NO option specified, the associated table spaces are placed in restore pending state. To remove the table spaces from restore pending state, a restore operation must be performed. A rollforward operation will only place a table space in the restore pending state if the load operation completed successfully.

Related concepts:

- "Pending States After a Load Operation" on page 235

Character Set and National Language Support

The DB2® UDB 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) may 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` modifier is specified, no initial comparison is performed, and an attempt is made to load the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is loaded.

Related reference:

- "LOAD" on page 133

Pending States After a Load Operation

The load utility uses *table* states to preserve database consistency during a load operation. These states can be checked by using the LOAD QUERY command.

The load and build phases of the load process place the target table in the load in progress table state. The load utility also places table spaces in the load in progress state when the COPY NO option is specified on a recoverable database. The table spaces remain in this state for the duration of the load operation and are returned to normal state if the transaction is committed or rolled back.

If the NO ACCESS option has been specified, the table cannot be accessed while the load is in progress. If the ALLOW READ ACCESS option has been specified, the data in the table that existed prior to the invocation of the load command will be available in read only mode during the load operation. If the ALLOW READ ACCESS option is specified and the load operation fails, the data that existed in the table prior to the load operation will continue to be available in read only mode after the failure.

To remove the load in progress table state (if the load operation has failed, or was interrupted), do one of the following:

- Restart the load operation. First, address the cause of the failure; for example, if the load utility ran out of disk space, add containers to the table space before attempting a load restart operation.
- Terminate the load operation.
- Invoke a LOAD REPLACE operation against the same table on which a load operation has failed.
- Recover table spaces for the loading table by using the RESTORE DATABASE command with the most recent table space or database backup, and then carry out further recovery actions.

Table spaces are placed in backup pending state if the load process completes, and:

- The database configuration parameter *logretain* is set to recovery, or *userexit* is enabled, and
- The load option COPY YES is not specified, and
- The load option NONRECOVERABLE is not specified.

The fourth possible state associated with the load process (check pending state) pertains to referential and check constraints, DATALINKS constraints, generated column constraints, materialized query computation, or staging table propagation. For example, if an existing table is a parent table containing a primary key referenced by a foreign key in a dependent table, replacing data in the parent table places both tables (not the table space) in check pending state. To validate a table for referential integrity and check constraints, issue the SET INTEGRITY statement after the load process completes, if the table has been left in check pending state.

Related concepts:

- “Checking for Integrity Violations” on page 123
- “Table Locking, Table States and Table Space States” on page 231

Related reference:

- “LIST TABLESPACES” in the *Command Reference*

Optimizing Load Performance

The performance of the load utility depends on the nature and the quantity of the data, the number of indexes, and the load options specified.

Unique indexes reduce load performance if duplicates are encountered. In most cases, it is still more efficient to create indexes during the load operation than to invoke the CREATE INDEX statement for each index after the load operation completes (see Figure 5).

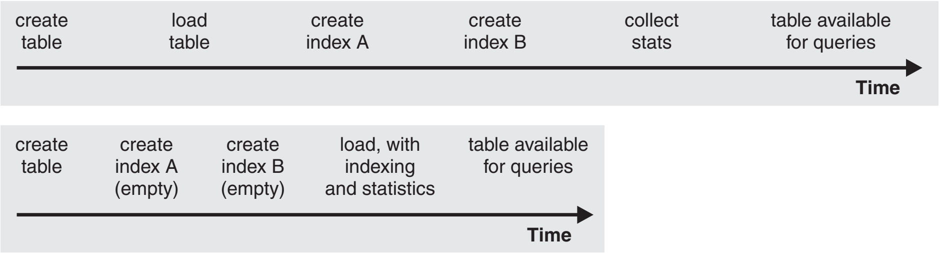


Figure 5. Increasing Load Performance through Concurrent Indexing and Statistics Collection. Tables are normally built in three steps: data loading, index building, and statistics collection. This causes multiple data I/O during the load operation, during index creation (there can be several indexes 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.

When tuning index creation performance, the amount of memory dedicated to the sorting of index keys during a load operation is controlled by the *sortheap* database configuration parameter. For example, to direct the load utility to use 4000 pages of main memory per index for key sorting, set the *sortheap* database configuration parameter to be 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 occurs. That is, the data is divided among several ″sort runs″ and stored in a temporary table space that will be merged later. If there is no way to avoid a sort spill by increasing the size of the *sortheap* parameter, it is important 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.

Load performance can be improved by installing high performance sorting libraries from third party vendors to create indexes during the load operation. An example of a third party sort product is SyncSort. Use the **DB2SORT** environment variable (registry value) to specify the location of the sorting library that is to be loaded at run time.

Use of the SET INTEGRITY statement may lengthen the total time needed to load a table and make it usable again. If all the load operations are performed in INSERT mode, the SET INTEGRITY statement will check the table for constraints violations incrementally (by checking only the appended portion of the table). If a table cannot be checked for constraints violations incrementally, the entire table is checked, and it may be some time before the table is usable again.

Similarly, if a load operation is performed on the underlying tables of a materialized query table, use of the REFRESH TABLE statement may lengthen the time needed to make both the underlying tables and the materialized query table fully usable again. If several sequential load operations are performed in INSERT mode into the underlying tables of a REFRESH IMMEDIATE materialized query table, the SET INTEGRITY statement will incrementally refresh the materialized query table in most cases. If the system determines that a full refresh is required, the materialized query table definition query will be recomputed, and it may be some time before the table is usable again.

The load utility performs equally well in INSERT mode and in REPLACE mode.

The 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 allowing the load utility to choose values for these parameters before attempting to tune them for your particular needs.

Following is information about the performance implications of various options available through the load utility:

ANYORDER

Specify this file type modifier to suspend the preservation of order in the data being loaded, and improve performance. If the data to be loaded is presorted, `anyorder` may corrupt the presorted order, and the benefits of presorting will be lost for subsequent queries.

BINARY NUMERICS and PACKED DECIMAL

Use these file type modifiers to improve performance when loading positional numeric ASC data into fixed-length records.

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 reduces load performance, because all of the loading data is copied during the load operation (forward recovery must be enabled); the increased I/O activity may 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 may reduce overall performance, because if forward recovery is enabled, the table is placed in backup pending state, and the database, or selected 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 6) regardless of the value of this parameter.

If tables include either LOB or LONG VARCHAR data, CPU_PARALLELISM is set to one. Parallelism is not supported in this case.

Although use of this parameter is not restricted to symmetric multiprocessor (SMP) hardware, you may not obtain any discernible performance benefit from using it in non-SMP environments.

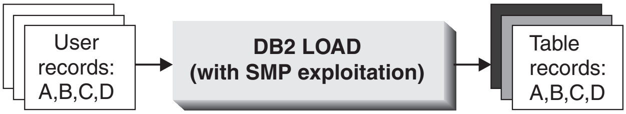


Figure 6. Record Order in the Source Data is Preserved When Intra-partition Parallelism is Exploited During a Load Operation

DATA BUFFER

The DATA BUFFER parameter specifies the total amount of memory allocated to the load utility as a buffer. It is recommended that this buffer be several *extents* in size. An extent is the unit of movement for data within DB2, and the extent size can be one or more 4KB pages. The DATA BUFFER parameter is useful when working with large objects (LOBs); it reduces I/O waiting time. 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* can be modified accordingly. The default value for the Utility Heap Size configuration parameter is 5 000 4KB 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.

FASTPARSE

Use the `fastparse` file type modifier to reduce the data checking that is performed on user-supplied column values, and enhance performance. This option should only be used when the data being loaded is known to be valid. It can improve performance by about 10 or 20 percent.

NONRECOVERABLE

Use this parameter if you do not need to be able to recover load transactions against a table. Load performance is enhanced, because no additional activity beyond the movement of data into the table is required, and the load operation completes without leaving the table spaces in backup pending state.

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 either be dropped or a LOAD TERMINATE command can be issued to bring it back online.

NOROWWARNINGS

Use the `norowwarnings` file type modifier to suppress the recording of warnings about rejected rows, and enhance performance, if you anticipate a large number of warnings.

ALLOW READ ACCESS

This option allows users to query a table while a load operation is in progress. The user will only be able to 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 may 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 will be used for I/O.

SAVECOUNT

Use this parameter to set an interval for the establishment of consistency points during a load operation. The synchronization of activities performed to establish a consistency point takes time. If done too frequently, there will be 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 ten million in the case of a load operation involving 100 million records).

A LOAD RESTART operation will automatically continue from the last consistency point.

STATISTICS YES

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 will decrease (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 DB2 BIND command.

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. For information about the UPDATE DATABASE CONFIGURATION command.

USE <tablespaceName>

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. When a load operation in ALLOW READ ACCESS mode fully rebuilds the indexes, the new indexes are built as a *shadow*. The original indexes are replaced by the new indexes at the end of the load operation.

By default, the *shadow* index is built in the same table space as the original index. This may cause resource problems as both the original and the *shadow* index will 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 will be instantaneously replaced by the *shadow*. However, if the *shadow* index is built in a system temporary table space, the load operation will require an index copy phase which will copy the index from a system temporary table space to the index table space. There will be 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 may not be sequential. The values specified by the DISK_PARALLELISM option will be 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. If you are expecting only a few warnings or no warnings, set the WARNINGCOUNT parameter to approximately the number you are expecting, or to twenty if you are expecting no warnings. The load operation will stop after the WARNINGCOUNT number is reached. This gives you the opportunity to correct data (or to drop and then recreate the table being loaded) before attempting to complete the load operation. Although not having a direct effect on the performance of the load operation, the establishment of a WARNINGCOUNT threshold prevents you from having to wait until the entire load operation completes before determining that there is a problem.

Related concepts:

- "DB2 registry and environment variables" in the *Administration Guide: Performance*

Related reference:

- "Utility Heap Size configuration parameter - util_heap_sz" in the *Administration Guide: Performance*
- "Statistics Heap Size configuration parameter - stat_heap_sz" in the *Administration Guide: Performance*
- "SET INTEGRITY statement" in the *SQL Reference, Volume 2*
- "BIND" in the *Command Reference*
- "UPDATE DATABASE CONFIGURATION" in the *Command Reference*

Load - 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)
```

Notes:

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 new-line character at the end of each input record, and that each record is 40 bytes long. The last 8 bytes are not used 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 DATALINK Data)

The following command loads the table MOVIETABLE from the input file `delfile1`, which has data in the DEL format:

```
db2 load from delfile1 of del
    modified by dldel|
    insert into movietable (actorname, description, url_making_of,
    url_movie) datalink specification (dl_url_default_prefix
    "http://narang"), (dl_url_replace_prefix "http://bomdel"
    dl_url_suffix ".mpeg") for exception excptab
```

Notes:

1. The table has four columns:

   ```
   actorname              VARCHAR(n)
   description            VARCHAR(m)
   url_making_of          DATALINK (with LINKTYPE URL)
   url_movie              DATALINK (with LINKTYPE URL)
   ```

2. The DATALINK data in the input file has the vertical bar (|) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, ″http://narang″ is used.
4. Each non-NULL column value for url_movie will get ″http://bomdel″ as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ″.mpeg″ appended to the path. For example, if a column value of url_movie is ″http://server1/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″; if the value is ″/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″.
6. If any unique index or DATALINK exception occurs while loading the table, the affected records are deleted from the table and put into the exception table `excptab`.

Example 4 (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
```

Notes:

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 5 (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)
```

Notes:
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.

Related concepts:
- "Load Overview" on page 102

Related reference:
- "LOAD QUERY" on page 178
- "Example Partitioned Database Load Sessions" on page 267
- "LOAD" on page 133

Chapter 4. Loading Data in a Partitioned Database Environment

This chapter describes loading data in a partitioned database environment.

The following topics are covered:

Loading Data in a Partitioned Database - Overview

In a partitioned database, large amounts of data are located across many partitions. Partitioning keys are used to determine on which database partition each portion of the data resides. The data must be *partitioned* before it can be loaded at the correct database partition. When loading tables in a partitioned database environment, the load utility can:

- Partition input data in parallel.
- Load data simultaneously on corresponding database partitions.
- Transfer data from one system to another system.

Partitioned database load operations take place in 2 phases: A setup phase, where partition resources such as table locks are acquired, and a load phase where the data is loaded into the partitions. You can use the ISOLATE_PART_ERRS option of the LOAD command to select how errors will be handled during either of these phases, and how errors on one or more of the partitions will affect the load operation on the partitions that are not experiencing errors.

When loading data into a partitioned database you can use one of the following modes:

- PARTITION_AND_LOAD. Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- PARTITION_ONLY. Data is partitioned (perhaps in parallel) and the output is written to files in a specified location on each loading partition. Each file includes a partition header that specifies how the data was partitioned, and that the file can be loaded into the database using the LOAD_ONLY mode.
- LOAD_ONLY. Data is assumed to be already partitioned; the partition process is skipped, and the data is loaded simultaneously on the corresponding database partitions.
- LOAD_ONLY_VERIFY_PART. Data is assumed to be already partitioned, but the data file does not contain a partition header. The partitioning 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 partition. Rows containing partition violations are placed in a dumpfile if the *dumpfile* file type modifier is specified. Otherwise, the rows are discarded. If partition violations exist on a particular loading partition, a single warning will be written to the load message file for that partition.
- ANALYZE. An optimal partitioning map with even distribution across all database partitions is generated.

Concepts and Terminology

The following terminology will be used when discussing the behavior and operation of the load utility in a partitioned database environment:

- The *coordinator partition* is the database partition to which the user connects 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 partition unless the CLIENT option of the load command is specified. Specifying the CLIENT option of the load command 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 will partition the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per partition for any load operation.
- In the PARTITION_AND_LOAD, LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, *load agents* run on each output partition and coordinate the loading of data to that partition.
- *Load to file agents* run on each output partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their partition.
- A *file transfer command agent* runs on the coordinator partition and is responsible for executing a file transfer command.

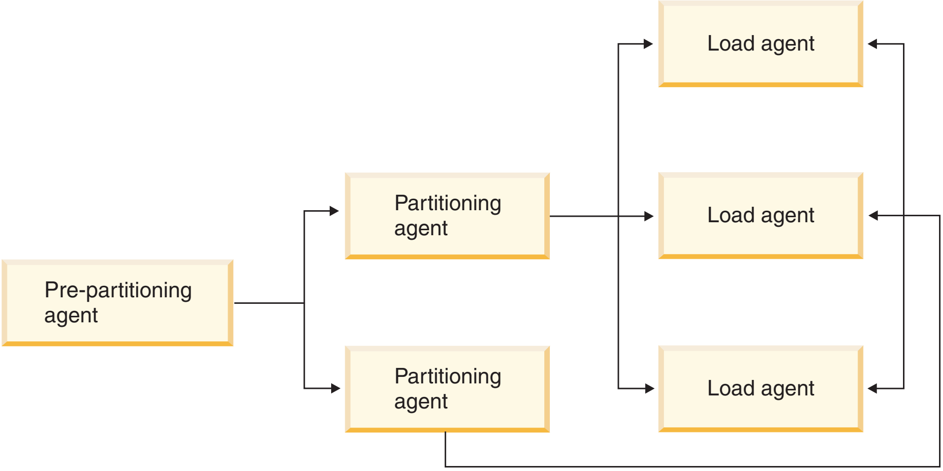


Figure 7. Partitioned Database Load Overview. The source data is read by the pre-partitioning agent, approximately half of the data is sent to each of two partitioning agents which partition the data and send it to one of three database partitions. The load agent at each partition loads the data.

Related concepts:

- "Restarting or Terminating a Load Operation in a Partitioned Database Environment" on page 260

Related reference:

- "Partitioned Database Load Configuration Options" on page 262

Using Load in a Partitioned Database Environment

Prerequisites:

Before loading a table in a partitioned database environment:

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 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 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.

Restrictions:

The following restrictions apply when using the load utility to load data in a partitioned database environment:

- 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 you are using multiple database partitions to partition and then load the data, the use of a SAVECOUNT greater than zero on the LOAD command is not 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 partition 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 partitions specified by the OUTPUT_DBPARTNUMS and PARTITIONING_DBPARTNUMS options do not overlap. For example, if a table is defined on partitions 0 through 3, one load operation can load data into partitions 0 and 1 while a second load operation can load data into partitions 2 and 3.
- Only Non-delimited ASCII (ASC) and Delimited ASCII (DEL) files can be partitioned. PC/IXF files cannot be partitioned. To load a PC/IXF file into a multiple partition table, you can first load it into a single-partition table, and then perform a load operation using the CURSOR file type to move the data into a multiple partition table.

Procedure:

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 partitions: 0, 1, 2, 3 and 4. Each partition has a local directory

/udb/data/. Two tables, TABLE1 and TABLE2, are defined on 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

Partition and Load Example

In this scenario you are connected to a partition that may or may not be a partition where TABLE1 is defined. The data file load.del resides in the current working directory of this partition. To load the data from load.del into all of the partitions where TABLE1 is defined, issue the following command: `LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1`

Note: In this example, default values will be used for all of the partitioned database configuration parameters: The MODE parameter will default to PARTITION_AND_LOAD, the OUTPUT_DBPARTNUMS options will default to all nodes on which TABLE1 is defined, and the PARTITIONING_DBPARTNUMS will default to the set of nodes selected according to the LOAD command rules for choosing partitioning nodes when none are specified.

To perform a load operation using nodes 3 and 4 as partitioning nodes, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                     PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
```

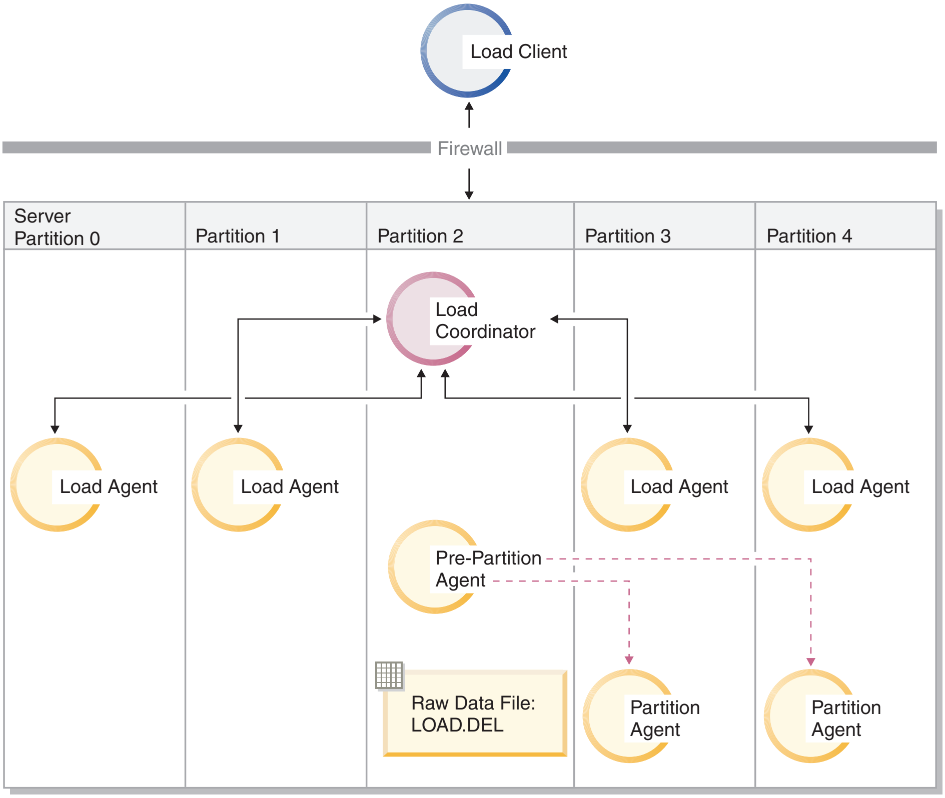


Figure 8. . This diagram illustrates the behavior that will result when the above command is issued. Data is loaded into partitions 3 and 4.

Partition Only Example

In this scenario you are connected to a partition that may or may not be a partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this partition. To partition (but not load) `load.del` to all the database partitions on which TABLE1 is defined, using partitions 3 and 4 as partitioning nodes, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                      PARTITIONED DB CONFIG MODE PARTITION_ONLY
                                          PART_FILE_LOCATION /udb/data
                                          PARTITIONING_DBPARTNUMS (3,4)
```

This will result in a file `load.del.xxx` being stored in the `/udb/data` directory on each partition, where `xxx` is a three-digit representation of the partition number.

To partition the `load.del` file to partitions 1 and 3, using only 1 partitioning agent running on 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 /udb/data
                                             OUTPUT_DBPARTNUMS (1,3)
```

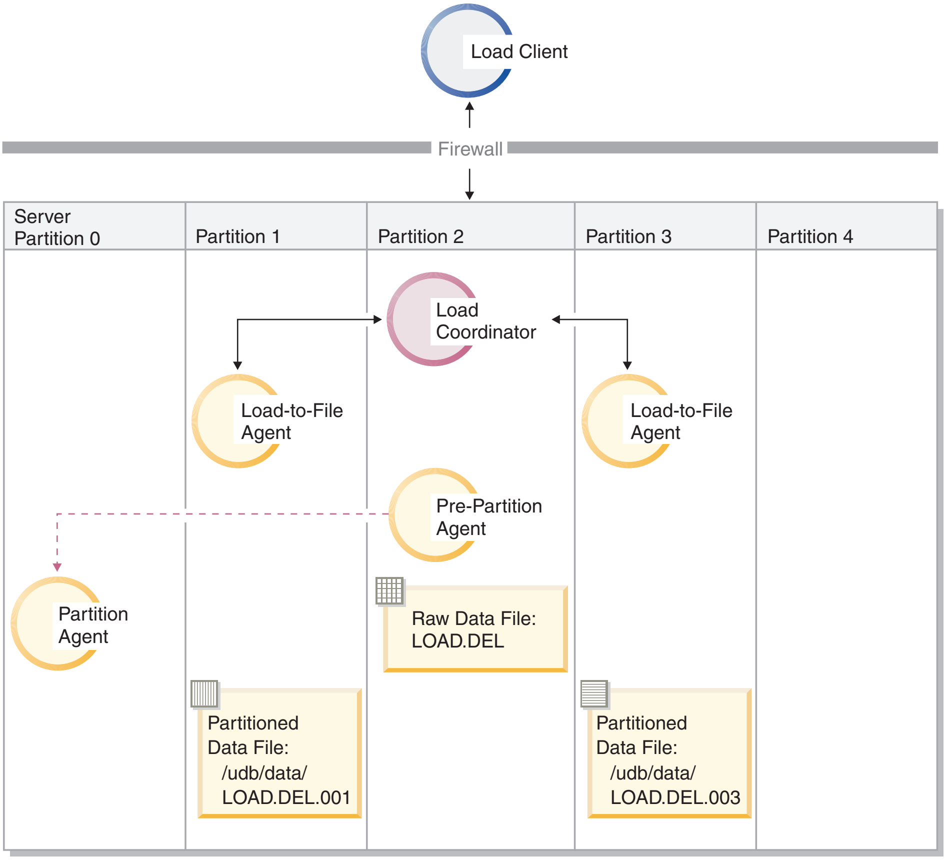
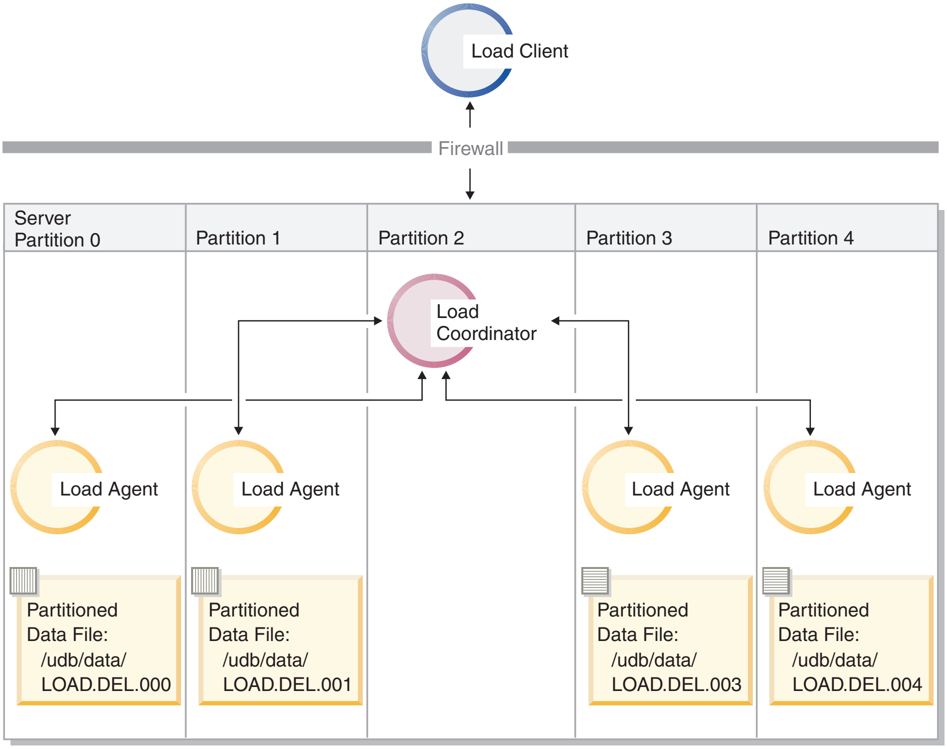


Figure 9. . This diagram illustrates the behavior that will result when the above command is issued. Data is loaded into partitions 1 and 3, using 1 partitioning agent running on partition 0.

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 `/udb/data` directory of each loading partition to all the 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 /udb/data
```



Load Client
Firewall
Server Partition 0
Partition 1
Partition 2
Partition 3
Partition 4
Load Coordinator
Load Agent
Load Agent
Load Agent
Load Agent
Partitioned Data File: /udb/data/ LOAD.DEL.000
Partitioned Data File: /udb/data/ LOAD.DEL.001
Partitioned Data File: /udb/data/ LOAD.DEL.003
Partitioned Data File: /udb/data/ LOAD.DEL.004

Figure 10. . This diagram illustrates the behavior that will result when the above command is issued. Partitioned data is loaded to all partitions where TABLE1 is defined.

To load into 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 /udb/data
                                                           OUTPUT_DBPARTNUMS (4)
```

Loading Pre-partitioned Files Without Partition Map Headers

The LOAD command can be used to load data files without partition headers directly into several database partitions. If the data files exist in the /udb/data directory on each partition where TABLE1 is defined and have the name load.del.xxx, where xxx is the 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 /udb/data
```

To load the data into 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 /udb/data
                                             OUTPUT_DBPARTNUMS (1)
```

Note: Rows that do not belong on the partition from which they were loaded will be rejected and put into the dumpfile, if one has been specified.

Loading from a Remote Client to a Partitioned Database

To load data into a partitioned 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 single partition databases, 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 will be the fully qualified base file name of the partitioned files that are created or loaded on each output partition. Multiple files may be created with the specified base name if there are LOB columns in the target table.

To partition all the rows in the answer set of the statement `SELECT * FROM TABLE1` to a file on each partition named `/udb/data/select.out.xxx` (where xxx is the node number), for future loading into TABLE2, issue the following commands from the DB2 Command Line Processor:

```
DECLARE C1 CURSOR FOR SELECT * FROM TABLE1

LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
                       PARTITIONED DB CONFIG MODE PARTITION_ONLY
                       PART_FILE_LOCATION /udb/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 /udb/data/select.out
```

Related concepts:

- "Moving Data Using the Cursor File Type" on page 302

Related reference:

- "db2Load - Load" on page 181

Monitoring a Partitioned Database Load Using Load Query

Message Files Produced During a Partitioned Database Load

During a load operation, message files are created by some of the load processes on the partitions where they are being executed. These files store all information, warning and error messages produced during the execution of the process. The load processes that produce message files that can be viewed by the user are the load agent, pre-partitioning agent and partitioning agent.

Users can connect to individual partitions during a load operation and issue the LOAD QUERY command against the target table. When issued from the CLP, this command will display the contents of all the message files that currently reside on that partition for the table that is specified in the LOAD QUERY command.

For example, table TABLE1 is defined on partitions 0 through 3 in database WSDB. A user is connected to partition 0 and issues the following LOAD command:

```
load from load.del of del replace into table1 partitioned db config
    partitioning_dbpartnums (1)
```

This command will initiate a load operation that includes load agents running on partitions 0, 1, 2 and 3; a partitioning agent running on partition 1; and a pre-partitioning agent running on partition 0.

Partition 0 will contain one message file for the pre-partitioning agent and one for the load agent on that 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
```

Partition 1 will contain 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 will 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 load API (**db2Load**), the messages option (piLocalMsgFileName) must be specified and the message files will be brought from the server to the client and stored for the user to view.

For partitioned database load operations initiated from the CLP, the message files will not be displayed to the console or retained. To save or view the contents of these files after a partitioned database load has completed, the MESSAGES option of the LOAD command must be specified. If this option is used, once the load operation has completed the message files on each partition will be transferred to the client machine and stored in files using the base name indicated by the MESSAGES option. For partitioned 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.<partition-number>
Partitioning Agent	<message-file-name>.PART.<partition-number>
Pre-partitioning Agent	<message-file-name>.PREP.<partition-number>

For example, if the MESSAGES option specifies `/wsdb/messages/load`, the load agent message file for partition 2 will be `/wsdb/messages/load.LOAD.002`.

Note: It is strongly recommended that the MESSAGES option be used for partitioned database load operations initiated from the CLP.

Restarting or Terminating a Load Operation in a Partitioned Database Environment

The load process in a partitioned database environment consists of two stages: the setup stage where partition-level resources such as table locks on output partitions are acquired, and the load stage where data is formatted and loaded into tables on the partitions. The four partition error isolation modes (LOAD_ERRS_ONLY, SETUP_ERRS_ONLY, SETUP_AND_LOAD_ERRS, and NO_ISOLATION) affect the behavior of load restart and terminate operations when there are errors during one or both of these stages. In general, if a failure occurs during the setup stage, restart and terminate operations are not necessary. However, a failure during the load stage will require a LOAD RESTART or a LOAD TERMINATE on all partitions involved in the load operation.

Failures During the Setup Stage

When a load operation fails on at least one partition during the setup stage and the setup stage errors are not being isolated (i.e., the error isolation mode is either LOAD_ERRS_ONLY or NO_ISOLATION), the entire load operation will be aborted and the state of the table on each partition will be rolled back to the state it was in prior to the load operation. In this case, there is no need to issue a LOAD RESTART or LOAD TERMINATE command.

When a load operation fails on at least one partition during the initial setup stage and setup stage errors are being isolated (i.e., the error isolation mode is either SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS), the load operation will continue on the partitions where the setup stage was successful, but the table on each of the failing partitions is rolled back to the state it was in prior to the load operation. In this case, there is no need to perform a load restart or terminate operation, unless there is also a failure during the load stage.

To complete the load process on the partitions where the load operation failed during the setup stage, issue a LOAD REPLACE or LOAD INSERT command and use the OUTPUT_DBPARTNUMS option to specify only the partition numbers of the partitions that failed during the original load operation.

For example, table TABLE1 is defined on partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load.del of del replace into table1 partitioned db config
   isolate_part_errs setup_and_load_errs
```

During the set up stage of the load operation there is a failure on partitions 1 and 3. Since setup stage errors are being isolated, the load operation will complete successfully and data will be loaded on partitions 0 and 2. To complete the load operation by loading data on partitions 1 and 3, issue the following command:

```
load from load.del of del replace into table1 partitioned db config
    output_dbpartnums (1, 3)
```

Failures During the Load Stage

If a load operation fails on at least one partition during the load stage of a partitioned database load operation, a LOAD RESTART or LOAD TERMINATE command must be issued on all partitions involved in the load operation whether or not they encountered an error while loading data. This is necessary because loading data in a partitioned database environment is done through a single transaction. If a load restart operation is initiated, loading will continue where it left off on all partitions.

For example, table TABLE1 is defined on partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load1.del of del replace into table1 partitioned db config
    isolate_part_errs no_isolation
```

During the load stage of the load operation there is a failure on partitions 1 and 3. To resume the load operation, the LOAD RESTART command must specify the same set of output partitions as the original command since the load operation must be restarted on all partitions:

```
load from load.del of del restart into table1 partitioned db config
    isolate_part_errs no_isolation
```

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.

If a LOAD TERMINATE command is used when a partitioned database load operation fails during the load stage, all work done in the previous load operation will be lost and the table on each partition will be returned to the state it was in prior to the initial load operation.

For example, table TABLE1 is defined on partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load.del of del replace into table1 partitioned db config
    isolate_part_errs no_isolation
```

If a failure occurs during the load stage, the load operation can be terminated by issuing a LOAD TERMINATE command that specifies the same output parameters as the original command:

```
load from load.del of del terminate into table1 partitioned db config
    isolate_part_errs no_isolation
```

Related concepts:

- "Loading Data in a Partitioned Database - Overview" on page 249

Partitioned Database Load Configuration Options

HOSTNAME X

This option is only relevant when used with the FILE_TRANSFER_CMD option. X is the name of the remote machine where the data file resides. This may be a z/OS host or another workstation. If this option is not specified, and the FILE_TRANSFER_CMD option is specified, the hostname *nohost* will be used.

FILE_TRANSFER_CMD X

Specifies a file executable, batch file, or script that will be called before data is loaded onto any partitions. The value specified must be a fully qualified path. The full path, including the execution file name, must not exceed 254 characters. The command will be invoked with the following syntax:

```
<COMMAND> <logpath> <hostname> <basepipename> <nummedia>
<source media list>
```

Where:

<COMMAND>

Is the command specified by the FILE:TRANSFER:CMD modifier.

<logpath>

Is the log path for the file from which the data is being loaded. Diagnostic or temporary data may be written to this path.

<hostname>

Is the value of the HOSTNAME option.

<basepipename>

Is the base name for named pipes that the load operation will create and expect to receive data from. One pipe is created for every source file on the LOAD command. Each of these files ends with .xxx, where xxx is the index of the source file for the LOAD command. For example, if there are 2 source files for the LOAD command, and the <basepipename> is pipe123, two named pipes would be created: `pipe123.000` and `pipe123.001`. The <COMMAND> file will populate these named pipes with user data.

<nummedia>
Specifies the number of media arguments which follow.

<source media lst>
Is the list of source files specified in the LOAD command. Each source file must be placed inside double quotation marks.

PART_FILE_LOCATION X
In PARTITION_ONLY mode, `X` is the fully qualified location of the partitioned files. In LOAD_ONLY mode, `X` is the fully qualified location of the partitioned files. This location must exist on each partition specified by the OUTPUT_DBPARTNUMS option. In PARTITION_ONLY mode, if this value is not specified an error will be returned. In LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, this parameter must be specified whenever the data file name specified in the LOAD command is not fully qualified.

Notes:

1. This parameter is not valid when the CLIENT parameter is specified and the load operation is taking place in the LOAD_ONLY or the LOAD_ONLY_VERIFY_PART modes.
2. For all file types other than CURSOR, the location refers to the path where the partitioned files exist or are to be created. For the CURSOR file type, 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 partition. Multiple files may be created with the specified base name if there are LOB columns in the target table.

OUTPUT_DBPARTNUMS X
`X` represents a list of partition numbers. The partition numbers represent the database partitions on which the load operation is to be performed. The partition numbers must be a subset of the database partitions on which the table is defined. The default is that all database partitions will be selected. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (e.g., (0, 2 to 10, 15)).

PARTITIONING_DBPARTNUMS X
`X` represents a list of partition numbers that will be used in the partitioning process. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (e.g., (0, 2 to 10, 15)). The partitioning nodes specified can be different

from the database partitions being loaded. If not specified, the LOAD command will determine how many partitions are needed and which partitions to use in order to achieve optimal performance.

If the ANYORDER modifier is not specified in the LOAD command, only one partition agent will be used in the load session. Further, if there is only one partition specified for the OUTPUT_DBPARTNUMS option, or the coordinator node of the load operation is not an element of OUTPUT:DBPARTNUMS, the coordinator node of the load operation is used as the partitioning partition. Otherwise, the first partition (not the coordinator node) in OUTPUT_DBPARTNUMS is used as the partitioning partition.

If the ANYORDER modifier is specified, the number of partitioning nodes is determined as follows: (number of partitions in OUTPUT_DBPARTNUMS)/4 + 1. Then, the number of partitioning nodes is chosen from the OUTPUT_DBPARTNUMS, excluding the partition being used to load the data.

MODE X

Specifies the mode in which the load operation will take place when loading a partitioned database. PARTITION_AND_LOAD is the default. Valid values are:

- PARTITION_AND_LOAD. Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- PARTITION_ONLY. Data is partitioned (perhaps in parallel) and the output is written to files in a specified location on each loading partition.
- LOAD_ONLY. Data is assumed to be already partitioned; the partition process is skipped, and the data is loaded simultaneously on the corresponding database partitions.
- LOAD_ONLY_VERIFY_PART. Data is assumed to be already partitioned, but the data file does not contain a partition header. The partitioning 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 partition. Rows containing partition violations are placed in a dumpfile if the *dumpfile* file type modifier is specified. Otherwise, the rows are discarded. If partition violations exist on a particular loading partition, a single warning will be written to the load message file for that partition.
- ANALYZE. An optimal partitioning map with even distribution across all database partitions is generated.

MAX_NUM_PART_AGENTS X

Specifies the maximum numbers of partitioning agents to be used in a load session. The default 25.

ISOLATE_PART_ERRS X

Indicates how the load operation will react to errors that occur on individual partitions. The default is LOAD_ERRS_ONLY. Valid values are:

- SETUP_ERRS_ONLY. Errors that occur on a partition during setup, such as problems accessing a partition, or problems accessing a table space or table on a partition, will cause the load operation to stop on the failing partitions but to continue on the remaining partitions. Errors that occur on a partition while data is being loaded will cause the entire operation to fail and roll back to the last point of consistency on each partition.
- LOAD_ERRS_ONLY. Errors that occur on a partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded the partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining partitions until a failure occurs or until all the data is loaded. On the 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 partitions the transaction will be aborted. Data on all of the partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the partitions where load operation completed and resume the load operation on partitions that experienced an error.

  Note: This mode cannot be used with the ALLOW READ ACCESS or COPY YES options of the LOAD command.
- SETUP_AND_LOAD_ERRS. In this mode, partition-level errors during setup or loading data cause processing to stop only on the affected partitions. As with the LOAD_ERRS_ONLY mode, when partition errors do occur while data is being loaded, the data on all partitions will remain invisible until a load restart operation is performed.

  Note: This mode cannot be used with the ALLOW READ ACCESS or COPY YES options of the LOAD command.
- NO_ISOLATION. Any error during the load operation causes the transaction to fail.

STATUS_INTERVAL X

X represents how often you will be 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

Specifies 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 DB2 registry variable will replace 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 will not be checked for truncation at input/output.

MAP_FILE_INPUT X

`X` specifies the input file name for the partitioning map. This parameter must be specified if the partitioning map is customized, as it points to the file containing the customized partitioning map. A customized partitioning 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 MPP Loader 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 partitioning map. This parameter should be used when the ANALYZE mode is specified. An optimal partitioning map with even distribution across all database partitions is generated. If this modifier 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 parameter of the LOAD command is specified. When this option is specified, each record will be checked for a new line character. The record length, as specified in the RecLen parameter, will also be checked.

DISTFILE X

Specifies the name of the partition distribution file. The default is DISTFILE.

OMIT_HEADER

Specifies that a partition map header should not be included in the partition file. If not specified, a header will be generated.

RUN_STAT_DBPARTNUM X

If the STATISTICS YES parameter has been specified in the LOAD command, statistics will be 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 will be collected on the first database partition in output partition list.

Related tasks:

- "Using Load in a Partitioned Database Environment" on page 251

Related reference:

- "REDISTRIBUTE DATABASE PARTITION GROUP" in the *Command Reference*
- "LOAD" on page 133

Example Partitioned Database Load Sessions

In the following examples, the database has 4 partitions numbered 0 through 4. Database WSDB is defined on all of the partitions, and table TABLE1 resides in the default node group which is also defined on all of the partitions.

Example 1

To load data into TABLE1 from the user data file `load.del` which resides on partition 0, connect to 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 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 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 partitioned 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 partitioned output files will be stored on each of the output partitions in the directory /udb/data:

```
load from load.del of del replace into table1 partitioned db config mode
   partition_only part_file_location /udb/data
```

The output from the load command will be 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 node, and these agents ran successfully. There was a pre-partitioning agent on the coordinator partition, and a partitioning agent running on node 1. The statistical summary indicates that 100,000 rows were successfully read by the pre-partitioning agent and 100,000 rows were successfully partitioned 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 /udb/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 partition ran successfully and that the sum of the number of rows loaded by all load agents is 100,000. No summary of rows partitioned is indicated since partitioning 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 partitions runs out of space in the table space during the load operation, the following output will 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 partition summary indicates that partition 1 ran out of space. Since the default error isolation mode is NO_ISOLATION. the load operation will be aborted on all partitions and a load restart or load terminate operation must be invoked. If additional space is added to the containers of the table space on partition 1, the load operation can be restarted as follows:

```
load from load.del of del restart into table1
```

Related tasks:

- "Using Load in a Partitioned Database Environment" on page 251

Related reference:

- “LOAD” on page 133

Migration and Back-level Compatibility

Using the Load Command in a Partitioned Database Environment

In Version 8 changes have been made to the LOAD command that will replace the functionality of the **db2atld** utility. As a result, you will notice some differences in the terminology being used.

The **db2atld** utility referred to a *splitter* and a *split file* (the output of the splitter). The equivalent when using the LOAD command is the *partitioning agent* and the *partition file*. Also, when using the LOAD command we refer to the data file being *partitioned* **not** *split*. The load modes and configuration options are named to reflect this change. For example, where the AutoLoader had the SPLIT_ONLY mode and the SPLIT_NODES option, the LOAD command now has the PARTITION_ONLY mode and the PARTITIONING_DBPARTNUMS option.

The equivalent of the FORCE option of the AutoLoader is the ISOLATE_PART_ERRS load configuration option. This option allows users to specify four types of partition error isolation. One of them (SETUP_AND_LOAD_ERRS) is similar to the error isolation offered by the FORCE YES option.

Using db2atld

The **db2atld** utility is still available and it accepts a configuration file with the same syntax it did prior to Version 8. Consider the following when using **db2atld**:

1. **db2atld** will now only establish one connection when loading into multi-partition databases. Prior to Version 8, the **db2atld** established one connection per output partition.
2. **db2atld** will use the same, single transaction model as the load utility when used in a partitioned database environment.
3. The LOAD_ONLY_VERIFY_PART mode and any other new features provided in the enhanced load utility are not available when initiating a partitioned database load via the **db2atld** executable.
4. All previously supported AutoLoader configuration parameters are recognized.
5. Information displayed on the console by **db2atld** is now slightly different. For example, in SPLIT_AND_LOAD mode the partitioning statistics and loading statistics are now always displayed. Previously, they were only displayed when there was a discrepancy between the statistics. Further, status interval information is not dumped to the console during the load

operation but is instead dumped to the pre-partitioning agent message file, which is called ″<messagefile>.PREP.<nodenum>.″ Status information regarding the spawning of the splitter processes and loading processes is no longer available.

6. It is no longer necessary to launch the **db2atld** executable from a path that is cross mounted.

Reverting to pre-Version 8 Load behavior Using the DB2® PARTITIONEDLOAD_DEFAULT registry variable

Prior to Version 8, you could load a single partition of a multi-partition database using the load utility as long as the input data file contained partition header information that was valid for the partition being loaded. This header was usually created in the SPLIT_ONLY mode of the AutoLoader or through the **db2split** utility. The partition header would be verified by the load utility before the remaining data was loaded. For tables residing in a single partition database partition group in a EEE environment, a data file with no partition header information could be loaded as long as the `noheader` file type modifier was specified.

The behavior of the LOAD command has since changed. In a partitioned database environment, when no partitioned database load configuration options are specified, it is now assumed that the load operation will take place on all partitions on which the table is defined. The input file does not require a partition header, and the MODE option defaults to PARTITION_AND_LOAD. To load a single partition, the OUTPUT_DBPARTNUMS option must be specified.

It is possible to maintain the pre-Version 8 behavior of the LOAD command in a partitioned database environment. This would allow you to load a file with a valid partition header into a single database partition without specifying any extra partitioned database configuration options. You can do this by setting the value of the DB2_PARTITIONEDLOAD_DEFAULT registry variable to NO. You may want 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 partitioned file into partition 3 of a table that resides in a 4 partition nodegroup, 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
```

Loading Data in a Partitioned Database Environment - Hints and Tips

Following is some information to consider before loading a table in a partitioned database environment:

- Familiarize yourself with the partitioned 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 wish to maintain that order during the loading process, only one database partition should be used for partitioning. Parallel partitioning cannot guarantee that the data will be loaded in the same order it was received. The load utility will choose 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 partitioned database job to continue even if the load operation detects (at startup time) that some loading 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 partitioned load configuration option to monitor the progress of a partitioned database load job. 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 partitioning nodes (as defined by the PARTITIONING_DBPARTNUMS option) are different from the loading partitions (as defined by the OUTPUT_DBPARTNUMS option), since there is less contention for CPU cycles. When loading data into a partitioned database, the load utility itself should be invoked on a database partition that is not participating in either the partitioning or the loading operation.
- Specifying the MESSAGES parameter in the LOAD command will save 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 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 partitioned load database configuration option can be used to specify that partition.

Troubleshooting

If the load utility is hanging, you can:

- Use the STATUS_INTERVAL parameter to monitor the progress of a partitioned 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 partitioning 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.

Chapter 5. Moving DB2 Data Links Manager Data

This chapter describes how to use the DB2 export, import, and load utilities to move DB2 Data Links Manager data.

For information about the file formats that you can use with these utilities, see "Export/Import/Load Utility File Formats" on page 323.

The following topics are covered:

- "Moving DB2 Data Links Manager Using Export - Concepts"
- "Using Export to Move DB2 Data Links Manager Data" on page 279
- "Using Import to Move DB2 Data Links Manager Data" on page 279
- "Using Load to Move DB2 Data Links Manager Data" on page 281.

Moving DB2 Data Links Manager Using Export - Concepts

Since table data resides in the database, and the files referred to by DATALINK columns reside on Data Links servers, the export utility must move both the database data, and the data files on the corresponding Data Links servers. To do this, the export utility produces one control file per Data Links server. The name of the control file is the same as the name of the Data Links server. The control files are created in a new directory that has the name `dlfm/`*YYYYMMDD/HHMMSS*, where *YYYYMMDD* represents *YearMonthDay*, and *HHMMSS* represents *HourMinuteSecond*. This directory is created under the same directory in which the export data file is created. A control file lists the names of the corresponding DB2 Data Links Manager files that are referenced by the DATALINK columns of the rows that are exported.

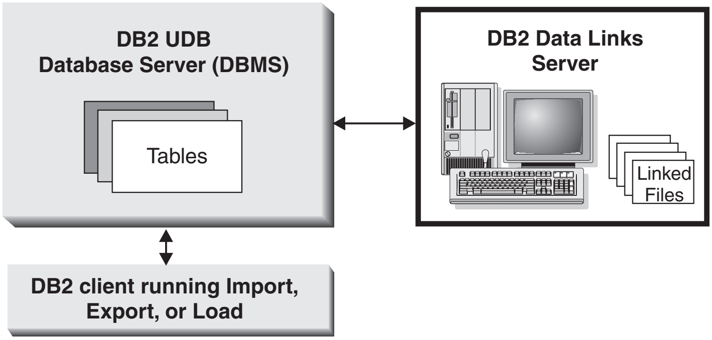


Figure 11. Moving DB2 Data Links Manager Data. Since table data resides in the database, and the files referred to by DATALINK columns reside on Data Links servers, the export, import, and load utilities must move both the database data, and the data files on the corresponding Data Links servers.

For the DCE Distributed File System (DFS) on AIX, the export utility produces one control file for all Data Links servers within a DCE cell. There will be one control file for each DCE cell. The name of the control file is the same as the name of the cell. The control file lists the URLs of all DB2 Data Links Manager files within a cell that are referenced by the DATALINK columns of the rows that are exported.

On Windows® operating systems, the export utility produces only one control file for all Data Links servers. The name of this control file is `ctrlfile.lst`. It is created in a new directory that has the name `dlfm\`*YYYYMMDD**HHMMSS*. This directory is created under the same directory in which the export data file is created. The control file lists the URLs of all DB2 Data Links Manager files that are referenced by the DATALINK columns of the rows that are exported.

DATALINK values that have the NO LINK CONTROL property are not placed in the control file.

The control files must be transported to their respective Data Links servers. For DFS, the control file for each cell must be transported to one of the Data Links servers within the cell. On the Windows NT® operating system, the single control file must be transported to all referenced Data Links servers. The **dlfm_export** utility should be run at each Data Links server, specifying the control file name. This utility produces an archive of the files listed in the control file for that Data Links server. For DFS, the **dlfm_export** utility should be run on the Data Links server to which the control file was transported. This utility produces an archive of the files listed in the control file for all the Data Links servers within the cell. Set the **DLFM_FS_ENVIRONMENT** registry variable appropriately before running the **dlfm_export** utility.

The export utility executes as an SQL application. The rows and columns that satisfy the conditions of the SELECT statement are extracted from the database. For DATALINK columns, the SELECT statement should not specify any scalar functions.

The export utility generates the following files:

- The export data file. A DATALINK column value in this file has the same format as that used by the import and the load utilities. If the DATALINK column value is NULL, it is treated in the same way as are other NULL columns.
- Control files for each Data Links server. The control file lists the complete path and the names of all the files that are to be exported from that Data Links server. For DFS, there will be one control file for each cell. On the Windows NT operating system, there is only one control file for all Data Links servers referenced by DATALINK column values.

Use the **dlfm_export** utility to export files from one or more Data Links servers as follows:

```
dlfm_export control-file-name archive-file-name
```

where *control-file-name* is the name of the control file generated by running the export utility on the DB2® client, and *archive-file-name* is the name of the archive file that will be generated. The default *archive-file-name* is `export.tar`, located in the current working directory.

A complementary utility called **dlfm_import** is provided to retrieve and restore files from the archive that **dlfm_export** generates. This utility must be used whether the archived files are being restored on the same, or a different, Data Links server.

Use the **dlfm_import** utility to retrieve files from the archive as follows:

```
dlfm_import archive-file-name [LISTFILES]
```

where *archive-file-name* is the name of the archive file that will be used to restore the files. The default *archive-file-name* is `export.tar`. LISTFILES is an optional keyword that, when specified, causes the utility to return a list of the files contained within the archive. Run the **dlfm_import** utility with root authority at each Data Links server, because you may want to restore the archived files on a different Data Links server, which may not have the same directory structure and user IDs as the Data Links server on which the **dlfm_export** utility was run. For DFS, run the **dlfm_import** utility at one of the Data Links servers within the target cell. This utility will get the DCE network root credentials before extracting the files from the archive. Set the **DLFM_FS_ENVIRONMENT** registry variable appropriately before running the **dlfm_import** utility.

Note: When running the **dlfm_import** utility on a Data Links server other than the one on which the **dlfm_export** utility was run, the files will be restored in the correct paths. The files will be owned by root in case some of the user IDs do not exist on the importing machine. Before inserting these files into a database, ensure that all files have the correct permissions and belong to the correct user IDs. For DFS, create the required file sets, and set up the required ACLs before running the **dlfm_import** utility. This utility will create the directory (if one does not exist) required to extract the files.

The following table shows (for the non-DFS case) how to export the DB2 data and the files that are referenced by the database called `SystemA` to the database called `SystemB`. SystemA uses the Data Links servers DLFM1 and DLFM2. SystemB uses the Data Links servers DLFMX and DLFMY. The files on DLFM1 will be exported to DLFMX, and the files on DLFM2 will be exported to DLFMY.

Database SystemA with Data Links Servers DLFM1 and DLFM2			**Step**
DB2 data on File	File1 with file names for DLFM1	File2 with file names for DLFM2	1) Run the dlfm_export command (as root) on both Data Links servers. This will produce an archive on both Data Links servers.
Database SystemB with Data Links Servers DLFMX and DLFMY			
	On DLFMX, restore from archive	On DLFMY, restore from archive	2) Run dlfm_import (as root) on both Data Links servers.
			3) Run the IMPORT command on SystemB, using the parameter DL_URL_REPLACE_ PREFIX to specify the appropriate Data Links server for each exported file.
When you run the IMPORT command on SystemB, the SystemA data and all files referenced by DATALINK columns are imported.			

Related tasks:

- "Using Export" on page 3
- "Using Export to Move DB2 Data Links Manager Data" on page 279

Related reference:

- "sqluexpr - Export" on page 17

Using Export to Move DB2 Data Links Manager Data

Procedure:

To ensure that a consistent copy of the table and the corresponding files that are referenced by the DATALINK columns are copied, perform the following steps:

1. Ensure that no update transactions are in progress when the export operation is running by issuing the following command:

   ```
   db2 quiesce tablespaces for table tablename share
   ```
2. Invoke the export utility.
3. Run the **dlfm_export** utility with root authority at each Data Links server; this will successfully archive files to which the Data Links File Manager administrator may not have access. For DFS, the **dlfm_export** utility will get the DCE network root credentials before archiving the files listed in the control file. The utility does not capture the ACLs information of the files that are archived. As input to **dlfm_export**, specify the name of the control file that was generated by the export utility.
4. Make the table available for updates by issuing the following command:

   ```
   db2 quiesce tablespaces for table tablename reset
   ```

Related concepts:

- "Export Overview" on page 1
- "Moving DB2 Data Links Manager Using Export - Concepts" on page 275

Related tasks:

- "Using Import to Move DB2 Data Links Manager Data" on page 279
- "Using Load to Move DB2 Data Links Manager Data" on page 281

Using Import to Move DB2 Data Links Manager Data

Since table data resides in the database, and the files referred to by DATALINK columns reside on Data Links servers, the import utility must move both the database data, and the data files on the corresponding Data Links servers.

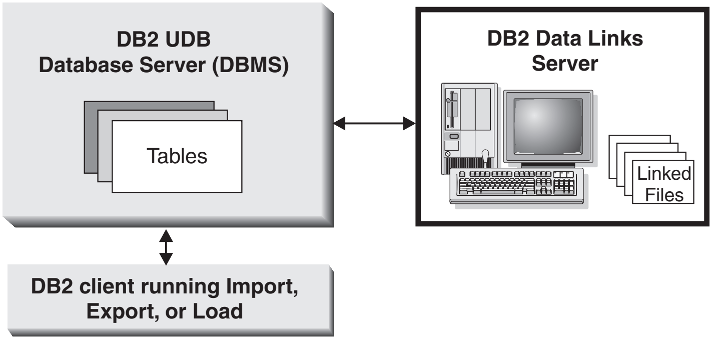


Figure 12. Moving DB2 Data Links Manager Data. Since table data resides in the database, and the files referred to by DATALINK columns reside on Data Links servers, the export, import, and load utilities must move both the database data, and the data files on the corresponding Data Links servers.

Procedure:

Before running the import utility against the target database:

1. Copy the files that will be referenced to the appropriate Data Links servers. The **dlfm_import** utility can be used to extract files from an archive that is generated by the **dlfm_export** utility.
2. Define the prefix name (or names) to the Data Links File Managers on the Data Links servers. (You may want to perform other administrative tasks, such as registering the database.)
3. Update the Data Links server information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required. (If the Data Links servers of the original configuration are the same as those at the target location, the Data Links server names need not be updated). For DFS, update the cell name information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required.
4. Define the Data Links servers at the target configuration in the DB2 Data Links Manager configuration file. For DFS, define the cells at the target configuration in the DB2 Data Links Manager configuration file. Register the DB2 Data Links Managers within the target cell using the **dlfm server_conf** command and the **dlfm client_conf** command.

When the import utility runs against the target database, files referred to by DATALINK column data are linked on the appropriate Data Links servers.

Related concepts:

- "Import Overview" on page 32

Related tasks:

- "Using Export to Move DB2 Data Links Manager Data" on page 279

- "Using Load to Move DB2 Data Links Manager Data" on page 281

Using Load to Move DB2 Data Links Manager Data

Procedure:

If you are loading data into a table with a DATALINK column that is defined with FILE LINK CONTROL, perform the following steps before invoking the load utility. (If all the DATALINK columns are defined with NO LINK CONTROL, these steps are not necessary.)

1. Ensure that DB2 Data Links Manager is installed on the Data Links servers that will be referred to by the DATALINK column values. For DFS, ensure that the DB2 Data Links Managers within the target cell are registered.
2. Ensure that the database is registered with the DB2 Data Links Manager.
3. Copy to the appropriate Data Links servers all files that will be inserted as DATALINK values.
4. Define the prefix name (or names) to the DB2 Data Links Managers on the Data Links servers.
5. Register the Data Links servers referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file. For DFS, register the cells at the target configuration referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file.

Links that fail during the load operation are considered to be data integrity violations, and are handled in much the same way as unique index violations. Consequently, a special exception has been defined for loading tables that have one or more DATALINK columns.

Following are load utility features that are not supported when loading tables with DATALINK columns:

- CPU_PARALLELISM (the value is forced to 1)
- LOAD REPLACE
- LOAD TERMINATE
- LOAD COPY

Related reference:

- "LOAD" on page 133

Chapter 6. Moving Data Between Systems

This chapter describes how to use the DB2 export, import, and load utilities to transfer data across platforms, and to and from iSeries host databases. DB2 DataPropagator, another method for moving data between databases in an enterprise, is also described. The chapter also introduces the Data Warehouse Center (DWC), which you can use to move data from operational databases to a warehouse database.

The following topics are covered:

- "Moving Data Across Platforms - File Format Considerations"
- "Moving Data With DB2 Connect" on page 285
- "db2move - Database Movement Tool" on page 287
- "db2relocatedb - Relocate Database" on page 292
- "Moving Data Between Typed Tables" on page 293
- "Using Replication to Move Data" on page 298
- "Using the Data Warehouse Center to Move Data" on page 300.
- "Moving Data Using the Cursor File Type" on page 302.

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.

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 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 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.

Related reference:

- "Export/Import/Load Utility File Formats" on page 323

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 from the host to the workstation, as well as the reverse (see Figure 13).

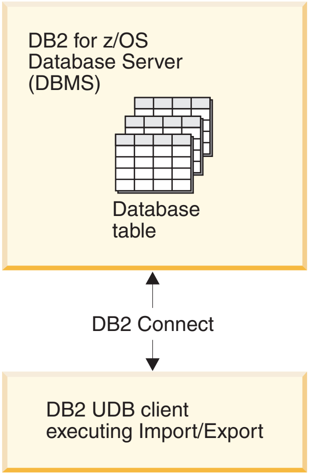


Figure 13. Import/Export through DB2 Connect

The DB2 export and import utilities allow you to move data from a host or AS/400 and iSeries 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 DB2 for MVS/ESA into a delimited ASCII 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.

Notes:

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 SQL. Specify the `compound` file type modifier in the import utility to group a specified number of SQL statements into a block. This may reduce network overhead and improve response time.

Restrictions:

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 to get the attributes of the source table. Import through DB2 Connect cannot create a table, because INSERT is the only supported option.
- A commit count interval must not be specified for the import operation.

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 AS/400 and iSeries 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 AS/400 and iSeries server database.

1. Export the data into an external IXF format by issuing the following command:

   ```
   db2 export to staff.ixf of ixf select * from userid.staff
   ```

2. Issue the following command to establish a DRDA connection to the target DB2 UDB server:

   ```
   db2 connect to cbc664 user admin using xxx
   ```

3. If it doesn't already exit, create the target table on target DB2 UDB server_

   ```
   CREATE TABLE mydb.staff
   ```

4. To import the data issue the following command:

   ```
   db2 import from staff.ixf of ixf insert into mydb.staff
   ```

   One 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 RELAVIS.STAFF. Single rows will continue to be inserted until all of the data has been moved to the target table.

Related concepts:

- "Moving Data Across Platforms - File Format Considerations" on page 283

Related reference:

- "EXPORT" on page 8
- "IMPORT" on page 42

db2move - Database Movement Tool

This tool 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.

Note: Tables with structured type columns are not moved when this tool is used.

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:

```
                                   ┌─────────────────────────┐
                                   ▼                         │
►►──db2move──dbname──action────────┬─────────────────────────┴──────────────────►◄
                                   ├─-tc──table-creators─────┤
                                   ├─-tn──table-names────────┤
                                   ├─-io──import-option──────┤
                                   ├─-lo──load-option────────┤
                                   ├─-l──lobpaths────────────┤
                                   ├─-u──userid──────────────┤
                                   └─-p──password────────────┘
```

Command parameters:

dbname
Name of the database.

action Must be one of: EXPORT, IMPORT, or LOAD.

-tc table-creators. The default is all creators.

This is an EXPORT action only. If specified, only those tables created by the creators listed with this option are exported. If not specified, the default is to use all creators. When specifying multiple creators, each must be separated by commas; no blanks are allowed between creator IDs. The maximum number of creators that can be specified is 10. 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 action only. If specified, only those tables whose names match exactly those in the specified string are exported. If not specified, the default is to use all user tables. When specifying multiple table names, each must be separated by commas; no blanks are allowed between table names. The maximum number of table names that can be specified is 10. This option can be used with the "-tc" table-creators option to select the tables for export. **db2move** will only export those tables whose names are matched with specified table names and whose creators are matched with specified table creators.

An asterisk (*) can be used as a wildcard character that can be placed anywhere in the string.

-io import-option. The default is REPLACE_CREATE.

Valid options are: INSERT, INSERT_UPDATE, REPLACE, CREATE, and REPLACE_CREATE.

-lo load-option. The default is INSERT.

Valid options are: INSERT and REPLACE.

-l lobpaths. The default is the current directory.

This option specifies the absolute path names where LOB files are created (as part of EXPORT) or searched for (as part of IMPORT or LOAD). When specifying multiple LOB paths, each must be separated by commas; no blanks are allowed between LOB paths. If the first path runs out of space (during EXPORT), or the files are not found in the path (during IMPORT or LOAD), the second path will be used, and so on.

If the action is EXPORT, and LOB paths are specified, all files in the LOB path directories are deleted, the directories are removed, and new directories are created. If not specified, the current directory is used for the LOB path.

-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. If the command is run on a client connecting to a remote server, user ID and password should be specified.

Examples:

- `db2move sample export`

  This will export all tables in the SAMPLE database; default values are used for all options.
- `db2move sample export -tc userid1,us*rid2 -tn tbname1,*tbname2`

  This will export all tables created by “userid1” or user IDs LIKE “us%rid2”, and with the name “tbname1” or table names LIKE “%tbname2”.
- `db2move sample import -l D:\LOBPATH1,C:\LOBPATH2`

  This example is applicable to the Windows operating system only. The command will import all tables in the SAMPLE database; LOB paths “D:\LOBPATH1” and “C:\LOBPATH2” are to be searched for LOB files.
- `db2move sample load -l /home/userid/lobpath,/tmp`

This example is applicable to UNIX based systems only. The command will load all tables in the SAMPLE database; both the `/home/userid/lobpath` subdirectory and the `tmp` subdirectory are to be searched for LOB files.

- `db2move sample import -io replace -u userid -p password`

  This will import all tables in the SAMPLE database in REPLACE mode; the specified user ID and password will be used.

Usage notes:

This tool exports, imports, or loads user-created tables. If a database is to be duplicated from one operating system to another operating system, **db2move** facilitates the movement of the tables. It is also necessary to move all other objects associated with the tables, such as: aliases, views, triggers, user-defined functions, and so on.

When export, import, or load APIs are called by **db2move**, the `FileTypeMod` parameter is set to `lobsinfile`. That is, LOB data is kept in separate files from PC/IXF files. There are 26 000 file names available for LOB files.

The LOAD action must be run locally on the machine where the database and the data file reside. When the load API is called by **db2move**, the `CopyTargetList` parameter is set to NULL; that is, no copying is done. If *logretain* is on, the load operation cannot be rolled forward later. The table space where the loaded tables reside is placed in backup pending state, and is not accessible. A full database backup, or a table space backup, is required to take the table space out of backup pending state.

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.

Related reference:

- "db2look - DB2 Statistics and DDL Extraction Tool" in the *Command Reference*

db2relocatedb - Relocate Database

Relocates a database as specified in the configuration file provided by the user.

Authorization:

None

Command syntax:

```
►►──db2relocatedb──-f──configFilename──────────────────────────────────────►◄
```

Command parameters:

-f configFilename
Specifies the name of the file containing configuration information necessary for relocating the database. This can be a relative or absolute filename.

Related reference:

- "db2inidb - Initialize a Mirrored Database" in the *Command Reference*

Moving Data Between Typed Tables

The DB2® export and import utilities can be used to move data out of, and into, typed tables. Typed tables may be in a hierarchy. Data movement across hierarchies can include:

- Movement from one hierarchy to an identical hierarchy.
- Movement from one hierarchy to a sub-section of a larger hierarchy.
- Movement from a sub-section of a large hierarchy to a separate hierarchy.

The IMPORT CREATE option allows you to create both the table hierarchy and the type hierarchy.

Identification of types in a hierarchy is database dependent. This means 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.

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. This order is called the *traverse order*. It is the order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the 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.

One method 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(s), then back up to next higher supertable, down to its subtables, and so on.

The following figure shows a hierarchy with four valid traverse orders:

- Person, Employee, Manager, Architect, Student.
- Person, Student, Employee, Manager, Architect (this traverse order is marked with the dotted line).
- Person, Employee, Architect, Manager, Student.
- Person, Student, Employee, Architect, Manager.

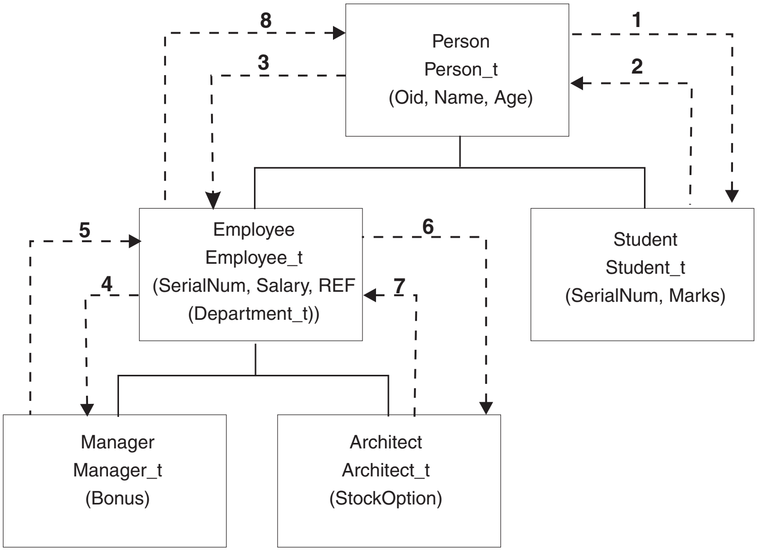


Figure 14.

Moving Data Between Typed Tables - Details

Traverse Order

There is a default traverse order, in which 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. There is also a user-specified traverse order, in which the user defines (in a traverse order list) the relevant types to be used. The same traverse order must be used when invoking the export utility and the import utility.

If you are specifying the traverse order, remember that the subtables must be traversed in PRE-ORDER fashion (that is, each branch in the hierarchy must be traversed to the bottom before a new branch is started).

Default Traverse Order

The default traverse order behaves differently when used with different file formats. Assume identical table hierarchy and type relationships in the following:

Exporting data to the PC/IXF file format creates a record of all relevant types, their definitions, and relevant tables. Export also completes the mapping of an index value to each table. During import, this mapping is used to ensure accurate movement of the data to the target database. 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 were created could be different, even though the source and target hierarchies may be structurally identical. This results in time differences that the default traverse order will identify when proceeding through the hierarchies. The creation time of each type determines the order taken 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

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. Given:

- 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

the import utility guarantees the accurate movement of data to the target database.

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 export and import utilities look for violations of this condition within the specified traverse order.

Selection During Data Movement

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. The export utility (in conjunction with the traverse order) controls what is placed in that file. You only need to specify the target table name and the WHERE clause. The export utility uses these selection criteria to create an appropriate intermediate file.

The import utility controls what is placed in the target database. You 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.

The import utility controls the size and the placement of the hierarchy being moved through the CREATE, INTO table-name, UNDER, and AS ROOT TABLE parameters.

Related reference:

- "IMPORT" on page 42

Examples of Moving Data Between Typed Tables

Examples in this section are based on the following hierarchical structure:

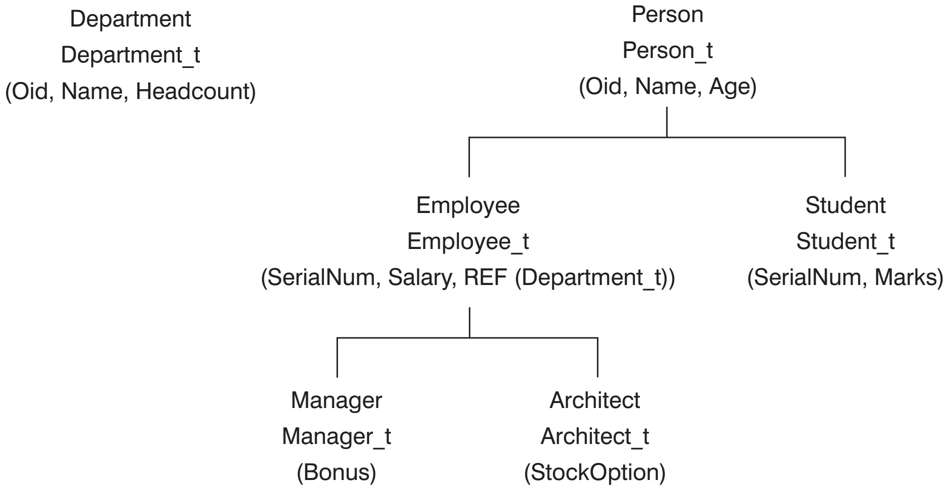


Figure 15.

Example 1

To export an entire hierarchy and then recreate it through an import operation:

```
DB2® CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.ixf OF IXF HIERARCHY STARTING Person
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 we are creating a new hierarchy, 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

In a more complex example, we would like to export the entire hierarchy of the source database and import it to the target database. Although we will export all of the data for those people over the age of 20, we will only import selected data to the target database:

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

- 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`) will not be 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` will also be 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

In this example, we 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 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.

Using Replication to Move Data

Replication allows you to copy data on a regular basis to multiple remote databases. If you need to have updates to a master database automatically copied to other databases, you can use the replication features of DB2® to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied. The replication features in DB2 are part of a larger IBM® solution for replicating data in small and large enterprises.

The IBM Replication tools are a set of DB2 DataPropagator™ programs and DB2 Universal Database™ tools that copy data between distributed relational database management systems:

- Between DB2 Universal Database platforms.
- Between DB2 Universal Database platforms and host databases supporting Distributed Relational Database Architecture™ (DRDA) connectivity.
- Between host databases that support DRDA® connectivity.

Data can also be replicated to non-IBM relational database management systems by way of DB2 DataJoiner.

You can use the IBM Replication tools to define, synchronize, automate, and manage copy operations from a single control point for data across your enterprise. The replication tools in DB2 Universal Database offer replication between relational databases. They also work in conjunction with IMS™ DataPropagator (formerly DPropNR) to replicate IMS and VSAM data, and with Lotus® NotesPump to replicate to and from Lotus Notes® databases.

Replication allows you to give end users and applications access to production data without putting extra load on the production database. You can copy the data to a database that is local to a user or an application, rather than have them access the data remotely. A typical replication scenario involves a source table with copies in one or more remote databases; for example, a central bank and its local branches. At predetermined times, automatic updates of the DB2 databases takes place, and all changes to the source database are copied to the target database tables.

The replication tools allow you to customize the copy table structure. You can use SQL when copying to the target database to enhance the data being copied. You can produce read-only copies that duplicate the source table, capture data at a specified point in time, provide a history of changes, or stage data to be copied to additional target tables. Moreover, you can create read-write copies that can be updated by end users or applications, and then have the changes replicated back to the master table. You can replicate views of source tables, or views of copies. Event-driven replication is also possible.

You can replicate data between DB2 databases on the following platforms: AIX, AS/400, HP-UX, Linux, Windows, OS/390, SCO UnixWare, Sun Solaris, Sequent, VM, and VSE. You can also replicate data between DB2 and the following non-DB2 databases (with DB2 DataJoiner): Informix, Microsoft® Jet, Microsoft SQL Server, Oracle, Sybase, and Sybase SQLAnywhere. In conjunction with other IBM products, you can replicate DB2 data to and from IMS, VSAM, or Lotus Notes. Finally, you can also replicate data to DB2 Everywhere on Windows® CE, or Palm OS devices.

IBM Replication Tools

The IBM Replication Tools by Component

There are two components of the IBM® Replication tools solution: the Capture program and the Apply program. You can set up these components using the DB2® Control Center. The operation and monitoring of these components happens outside of the Control Center.

The Capture program captures changes to the source tables. A source table can be:

- An external table containing SQL data from a file system or a nonrelational database manager loaded outside DB2 DataPropagator.
- An existing table in the database.
- A table that has previously been updated by the Apply program, which allows changes to be copied back to the source, or to other target tables.

The changes are copied into a change data table, where they are stored until the target system is ready to copy them. The Apply program then takes the changes from the change data table, and copies them to the target tables.

Use the Control Center to:

- Set up the replication environment.
- Define source and target tables.
- Specify the timing of automated copying.
- Specify SQL enhancements to the data.

- Define relationships between the source and the target tables.

Related tasks:

- "Planning for replication" in the *Replication Guide and Reference*

Using the Data Warehouse Center to Move Data

You can use the Data Warehouse Center (DWC) to move data from operational databases to a warehouse database, which users can query for decision support. You can also use the DWC to define the structure of the operational databases, called *sources*. You can then specify how the operational data is to be moved and transformed for the warehouse. You can model the structure of the tables in the warehouse database, called *targets*, or build the tables automatically as part of the process of defining the data movement operations.

The Data Warehouse Center uses the following DB2® functions to move and transform data:

- SQL

  You can use SQL to select data from sources and insert the data into targets. You also can use SQL to transform the data into its warehouse format. You can use the Data Warehouse Center to generate the SQL, or you can write your own SQL.

- Load and export utilities

  You can use these DB2 utilities to export data from a source, and then load the data into a target. These utilities are useful if you need to move large quantities of data. The Data Warehouse Center supports the following types of load and export operations:

  DB2 data export
  Exports data from a local DB2 database into a delimited file.

  ODBC data export
  Selects data from a table in a database that is registered to ODBC, and then writes the data to a delimited file.

  DB2 load
  Loads data from a delimited file into a DB2 table.

  DB2 load into a DB2 UDB ESE database (AIX only)
  Loads data from a delimited file into a DB2 Universal Database™ Enterprise Server Edition database, replacing existing data in a table with new data. This operation acquires the target partitioning map for the database, partitions the input file so that each file can be loaded on a database partition, and then runs a remote load operation on all partitions.

- Replication

  You also can use replication to copy large quantities of data from warehouse sources into a warehouse target, and then capture any subsequent changes to the source data. The Data Warehouse Center supports the following types of replication:

  Base aggregate
  : Creates a target table that contains aggregated data for a user table, and that is appended at specified intervals.

  Change aggregate
  : Creates a target table that contains aggregated data, and that is based on changes that are recorded for a source table.

  Point-in-time
  : Creates a target table that matches the source table, and adds a time stamp column to the target table.

  Staging table
  : Creates a ″consistent-change-data″ table that can be used as the source for updated data to multiple target tables.

  User copy
  : Creates a target table that matches the source table at the time that the copy is made.

  These operations are supported on all of the DB2 Universal Database workstation operating environments, DB2 Universal Database for OS/390, DB2 for AS/400, and DataJoiner.
- Transformer stored procedures

  You can use the Data Warehouse Center to move data into an OLAP (Online Analytical Processing) database. After the data is in the warehouse, you can use transformer stored procedures to clean up the data and then aggregate it into fact and dimension tables. You can also use the transformers to generate statistical data. Once the data is cleaned up and transformed, you can load it into OLAP cubes, or replicate it to departmental servers, which are sometimes called *datamarts*. The transformers are included in only some of the DB2 offerings. See your IBM® representative for more information.

Related concepts:

- “What is data warehousing?” in the *Data Warehouse Center Administration Guide*

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. By referencing a nickname within the SQL query, the load utility can also load data from another database in a single step.

To execute a load from cursor operation from the CLP, a cursor must first be declared against an SQL query. Once this is done, you can issue the LOAD command using the declared cursor's name as the *cursorname* and CURSOR as the file type.

For example:

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
```

Notes:

1. The above example shows how to load from an SQL query through the CLP. However, loading from an SQL query can also be done through the **db2Load** API, by properly defining the *piSourceList* and *piFileType* values of the *db2LoadStruct* structure.
2. As demonstrated above, 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.

Related concepts:

- "Nicknames and data source objects" in the *Federated Systems Guide*

Related reference:

- "LOAD" on page 133

- “Assignments and comparisons” in the *SQL Reference, Volume 1*

Appendix A. How to Read the Syntax Diagrams

A syntax diagram shows how a command should be specified so that the operating system can correctly interpret what is typed.

Read a syntax diagram from left to right, and from top to bottom, following the horizontal line (the main path). If the line ends with an arrowhead, the command syntax is continued, and the next line starts with an arrowhead. A vertical bar marks the end of the command syntax.

When typing information from a syntax diagram, be sure to include punctuation, such as quotation marks and equal signs.

Parameters are classified as keywords or variables:

- Keywords represent constants, and are shown in uppercase letters; at the command prompt, however, keywords can be entered in upper, lower, or mixed case. A command name is an example of a keyword.
- Variables represent names or values that are supplied by the user, and are shown in lowercase letters; at the command prompt, however, variables can be entered in upper, lower, or mixed case, unless case restrictions are explicitly stated. A file name is an example of a variable.

A parameter can be a combination of a keyword and a variable.

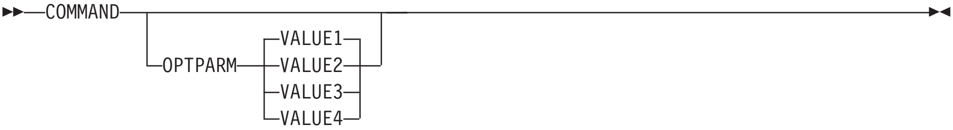
Required parameters are displayed on the main path:

```
►►──COMMAND──required parameter──────────────────────────────────────────►◄
```

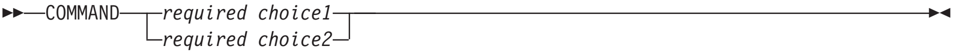
Optional parameters are displayed below the main path:

```
►►──COMMAND──┬────────────────────┬──────────────────────────────────────►◄
             └─optional parameter─┘
```

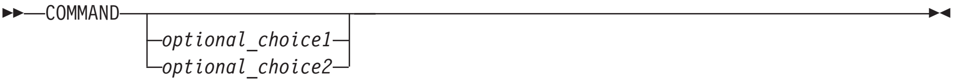
A parameter’s default value is displayed above the path:



A stack of parameters, with the first parameter displayed on the main path, indicates that one of the parameters must be selected:

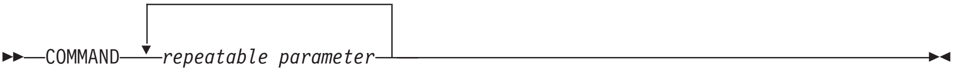


A stack of parameters, with the first parameter displayed below the main path, indicates that one of the parameters can be selected:

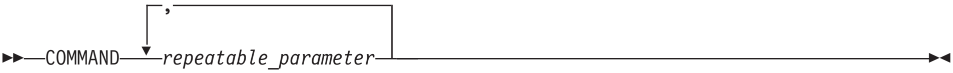


An arrow returning to the left, above the path, indicates that items can be repeated in accordance with the following conventions:

- If the arrow is uninterrupted, the item can be repeated in a list with the items separated by blank spaces:

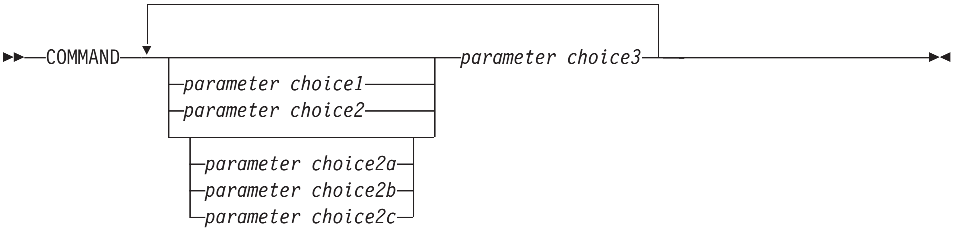


- If the arrow contains a comma, the item can be repeated in a list with the items separated by commas:

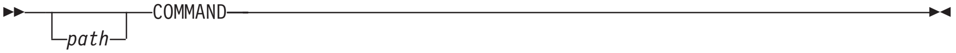


Items from parameter stacks can be repeated in accordance with the stack conventions for required and optional parameters discussed previously.

Some syntax diagrams contain parameter stacks within other parameter stacks. Items from stacks can only be repeated in accordance with the conventions discussed previously. That is, if an inner stack does not have a repeat arrow above it, but an outer stack does, only one parameter from the inner stack can be chosen and combined with any parameter from the outer stack, and that combination can be repeated. For example, the following diagram shows that one could combine parameter *choice2a* with parameter *choice2*, and then repeat that combination again (*choice2* plus *choice2a*):

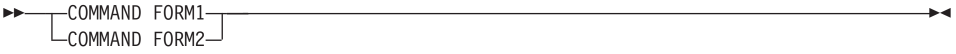


Some commands are preceded by an optional path parameter:



If this parameter is not supplied, the system searches the current directory for the command. If it cannot find the command, the system continues searching for the command in all the directories on the paths listed in the `.profile`.

Some commands have syntactical variants that are functionally equivalent:



Appendix B. 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.	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.
No CODEPAGE support.	CODEPAGE support, converting character data (and numeric data specified in characters) from this code page to the database code page during the load operation.
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 GENERATEDIGNORE and IDENTITYIGNORE file type modifiers.
Supports import into tables and views.	Supports loading into tables only.
All rows are logged.	Minimal logging is performed.
Trigger support.	No trigger support.

Import Utility	**Load Utility**
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.
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 reside on the node from which the import utility is invoked.	In a partitioned database environment, load files or pipes must reside on the node that contains the database. In a non-partitioned database environment, load files or pipes can reside on the node that contains the database, or on the remotely connected client from which the load utility is invoked.
A backup image is not required. Because the import utility uses SQL inserts, DB2 logs the activity, and no backups are required to recover these operations in case of failure.	A backup image can be created during the load operation.

Appendix C. Export/Import/Load Sessions - API Sample Program

The following sample program shows how to:

- Export data to a file
- Import data to a table
- Load data into a table
- Check the status of a load operation

The source file for this sample program (tbmove.sqc) can be found in the \sqllib\samples\c directory. It contains both DB2 APIs and embedded SQL calls. The script file bldapp.cmd, located in the same directory, contains the commands to build this and other sample programs.

To run the sample program (executable file), enter tbmove. You might find it useful to examine some of the generated files, such as the message file, and the delimited ASCII data file.

```
/****************************************************************************
** Licensed Materials - Property of IBM
** Governed under the terms of the IBM Public License
**
** (C) COPYRIGHT International Business Machines Corp. 2002
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*****************************************************************************
**
** SOURCE FILE NAME: tbmove.sqc
**
** SAMPLE: How to move table data
**
** DB2 APIs USED:
**         sqluexpr -- Export
**         sqluimpr -- Import
**         sqluvqdp -- Quiesce Table Spaces for Table
**         db2Load -- Load
**         db2LoadQuery -- Load Query
**
** SQL STATEMENTS USED:
**         PREPARE
**         DECLARE CURSOR
**         OPEN
**         FETCH
**         CLOSE
**         CREATE TABLE
**         DROP
**
** OUTPUT FILE: tbmove.out (available in the online documentation)
**
** For more information about the sample programs, see the README file.
**
** For more information about programming in C, see the
** "Programming in C and C++" section of the "Application Development Guide".
**
** For more information about building C applications, see the
** section for your compiler in the "Building Applications" chapter
** for your platform in the "Application Development Guide".
**
** For more information about SQL, see the "SQL Reference".
**
** For more information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, compiling, and running DB2
** applications, refer to the DB2 application development website at
**     http://www.software.ibm.com/data/db2/udb/ad
****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>
#include <db2ApiDf.h>
#include "utilemb.h"

int DataExport(char *);
int TbImport(char *);
int TbLoad(char *);
int TbLoadQuery(void);

/* support function */
int ExportedDataDisplay(char *);
int NewTableDisplay(void);

EXEC SQL BEGIN DECLARE SECTION;
  char strStmt[256];
  short deptnumb;
  char deptname[15];
EXEC SQL END DECLARE SECTION;

int main(int argc, char *argv[])
{
  int rc = 0;
  char dbAlias[SQL_ALIAS_SZ + 1];
  char user[USERID_SZ + 1];
  char pswd[PSWD_SZ + 1];
  char dataFileName[256];

  /* check the command line arguments */
  rc = CmdLineArgsCheck1(argc, argv, dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  printf("\nTHIS SAMPLE SHOWS HOW TO MOVE TABLE DATA.\n");

  /* connect to database */
  rc = DbConn(dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

#if(defined(DB2NT))
  sprintf(dataFileName, "%s%stbmove.DEL", getenv("DB2PATH"), PATH_SEP);
#else /* UNIX */
  sprintf(dataFileName, "%s%stbmove.DEL", getenv("HOME"), PATH_SEP);
#endif

  rc = DataExport(dataFileName);
  rc = TbImport(dataFileName);
  rc = TbLoad(dataFileName);
  rc = TbLoadQuery();

  /* disconnect from the database */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* main */

int ExportedDataDisplay(char *dataFileName)
{
  struct sqlca sqlca;
  FILE *fp;
  char buffer[100];
  int maxChars = 100;
  int numChars;
  int charNb;

  fp = fopen(dataFileName, "r");
  if (fp == NULL)
  {
    return 1;
  }

  printf("\n  The content of the file '%s' is:\n", dataFileName);
  printf("    ");
  numChars = fread(buffer, 1, maxChars, fp);
  while (numChars > 0)
  {
    for (charNb = 0; charNb < numChars; charNb++)
    {
      if (buffer[charNb] == '\n')
      {
        printf("\n");
        if (charNb < numChars - 1)
        {
          printf("    ");
        }
      }
      else
      {
        printf("%c", buffer[charNb]);
      }
    }
    numChars = fread(buffer, 1, maxChars, fp);
  }

  if (ferror(fp))
  {
    fclose(fp);
    return 1;
  }
  else
  {
    fclose(fp);
  }

  return 0;
} /* ExportedDataDisplay */

int NewTableDisplay(void)
{
  struct sqlca sqlca;

  printf("\n  SELECT * FROM newtable\n");
  printf("    DEPTNUMB DEPTNAME      \n");
  printf("    -------- --------------\n");

  strcpy(strStmt, "SELECT * FROM newtable");

  EXEC SQL PREPARE stmt FROM :strStmt;
  EMB_SQL_CHECK("statement -- prepare");

  EXEC SQL DECLARE c0 CURSOR FOR stmt;

  EXEC SQL OPEN c0;
  EMB_SQL_CHECK("cursor -- open");

  EXEC SQL FETCH c0 INTO :deptnumb, :deptname;
  EMB_SQL_CHECK("cursor -- fetch");

  while (sqlca.sqlcode != 100)
  {
    printf("    %8d %-s\n", deptnumb, deptname);

    EXEC SQL FETCH c0 INTO :deptnumb, :deptname;
    EMB_SQL_CHECK("cursor -- fetch");
  }

  EXEC SQL CLOSE c0;

  return 0;
} /* NewTableDisplay */

int DataExport(char *dataFileName)
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqldcol dataDescriptor;
  char actionString[256];
  struct sqlchar *pAction;
  char msgFileName[128];
  struct sqluexpt_out outputInfo;

  printf("\n-----------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  sqluexpr -- Export\n");
  printf("TO EXPORT DATA TO A FILE.\n");

  printf("\n  Be sure to complete all table operations and release\n");
  printf("  all locks before starting an export operation. This\n");
  printf("  can be done by issuing a COMMIT after closing all\n");
  printf("  cursors opened WITH HOLD, or by issuing a ROLLBACK.\n");
  printf("  Please refer to the 'Administrative API Reference'\n");
  printf("  for the details.\n");

  /* export data */
  dataDescriptor.dcolmeth = SQL_METH_D;
  strcpy(actionString, "SELECT deptnumb, deptname FROM org");
  pAction = (struct sqlchar *)malloc(sizeof(short) +
                                     sizeof(actionString) + 1);
  pAction->length = strlen(actionString);
  strcpy(pAction->data, actionString);
  strcpy(msgFileName, "tbexport.MSG");
  outputInfo.sizeOfStruct = SQLUEXPT_OUT_SIZE;

  printf("\n  Export data.\n");
  printf("    client destination file name: %s\n", dataFileName);
  printf("    action                      : %s\n", actionString);
  printf("    client message file name    : %s\n", msgFileName);

  /* export data */
  sqluexpr(dataFileName,
           NULL,
           NULL,
           &dataDescriptor,
           pAction,
           SQL_DEL,
           NULL,
           msgFileName,
           SQLU_INITIAL,
           &outputInfo,
           NULL,
           &sqlca);
  DB2_API_CHECK("data -- export");

  /* free memory allocated */
  free(pAction);

  /* display exported data */
  rc = ExportedDataDisplay(dataFileName);

  return 0;
} /* DataExport */

int TbImport(char *dataFileName)
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqldcol dataDescriptor;
  char actionString[256];
  struct sqlchar *pAction;
  char msgFileName[128];
  struct sqluimpt_in inputInfo;
  struct sqluimpt_out outputInfo;

  printf("\n------------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  sqluimpr -- Import\n");
  printf("TO IMPORT DATA TO A TABLE.\n");

  /* create new table */
  printf("\n  CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,");
  printf("\n                        deptname VARCHAR(14))\n");

  EXEC SQL CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,
                                 deptname VARCHAR(14));
  EMB_SQL_CHECK("new table -- create");

  /* import table */
  dataDescriptor.dcolmeth = SQL_METH_D;
  strcpy(actionString, "INSERT INTO newtable");
  pAction = (struct sqlchar *)malloc(sizeof(short) +
                                     sizeof(actionString) + 1);
  pAction->length = strlen(actionString);
  strcpy(pAction->data, actionString);
  strcpy(msgFileName, "tbimport.MSG");
  inputInfo.sizeOfStruct = SQLUIMPT_IN_SIZE;
  inputInfo.commitcnt = 10;
  inputInfo.restartcnt = 0;
  outputInfo.sizeOfStruct = SQLUIMPT_OUT_SIZE;
  printf("\n  Import table.\n");
  printf("    client source file name : %s\n", dataFileName);
  printf("    action                  : %s\n", actionString);
  printf("    client message file name: %s\n", msgFileName);

  /* import table */
  sqluimpr(dataFileName,
           NULL,
           &dataDescriptor,
           pAction,
           SQL_DEL,
           NULL,
           msgFileName,
           SQLU_INITIAL,
           &inputInfo,
           &outputInfo,
           NULL,
           NULL,
           &sqlca);
  DB2_API_CHECK("table -- import");

  /* free memory allocated */
  free(pAction);

  /* display import info */
  printf("\n  Import info.\n");
  printf("    rows read     : %ld\n", outputInfo.rowsRead);
  printf("    rows skipped  : %ld\n", outputInfo.rowsSkipped);
  printf("    rows inserted : %ld\n", outputInfo.rowsInserted);
  printf("    rows updated  : %ld\n", outputInfo.rowsUpdated);
  printf("    rows rejected : %ld\n", outputInfo.rowsRejected);
  printf("    rows committed: %ld\n", outputInfo.rowsCommitted);

  /* display content of the new table */
  rc = NewTableDisplay();

  /* drop new table */
  printf("\n  DROP TABLE newtable\n");

  EXEC SQL DROP TABLE newtable;
  EMB_SQL_CHECK("new table -- drop");

  return 0;
} /* TbImport */

int TbLoad(char *dataFileName)
{
  int rc = 0;
  struct sqlca sqlca;

  struct db2LoadStruct paramStruct;
  struct db2LoadIn inputInfoStruct;
  struct db2LoadOut outputInfoStruct;

  struct sqlu_media_list mediaList;
  struct sqldcol dataDescriptor;
  char actionString[256];
```

```
struct sqlchar *pAction;
char localMsgFileName[128];

printf("\n-----------------------------------------------------------");
printf("\nUSE THE DB2 API:\n");
printf("  sqluvqdp -- Quiesce Table Spaces for Table\n");
printf("  db2Load -- Load\n");
printf("TO LOAD DATA INTO A TABLE.\n");

/* create new table */
printf("\n  CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,");
printf("\n                        deptname VARCHAR(14))\n");

EXEC SQL CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,
                               deptname VARCHAR(14));
EMB_SQL_CHECK("new table -- create");

/* quiesce table spaces for table */
printf("\n  Quiesce the table spaces for 'newtable'.\n");

EXEC SQL COMMIT;
EMB_SQL_CHECK("transaction -- commit");

/* quiesce table spaces for table */
sqluvqdp("newtable", SQLU_QUIESCEMODE_RESET_OWNED, NULL, &sqlca);
DB2_API_CHECK("tablespaces for table -- quiesce");

/* load table */
mediaList.media_type = SQLU_CLIENT_LOCATION;
mediaList.sessions = 1;
mediaList.target.location =
  (struct sqlu_location_entry *)malloc(sizeof
       (struct sqlu_location_entry) * mediaList.sessions);
strcpy(mediaList.target.location->location_entry, dataFileName);

dataDescriptor.dcolmeth = SQL_METH_D;

strcpy(actionString, "INSERT INTO newtable");
pAction = (struct sqlchar *)malloc(sizeof(short) +
                                   sizeof(actionString) + 1);
pAction->length = strlen(actionString);
strcpy(pAction->data, actionString);

strcpy(localMsgFileName, "tbload.MSG");

/* Setup the input information structure */
inputInfoStruct.piUseTablespace       = NULL;
inputInfoStruct.iSavecount    =  0;       /* consistency points as
inputInfoStruct.iRestartcount =  0;       /* start at row 1 */
inputInfoStruct.iRowcount  =  0;       /* load all rows */
inputInfoStruct.iWarningcount =  0;        /* don't stop for warnings */
inputInfoStruct.iDataBufferSize =  0;      /* default data buffer size */
inputInfoStruct.iSortBufferSize  =  0;     /* def. warning buffer size */
inputInfoStruct.iHoldQuiesce     =  0;     /* don't hold the quiesce */
inputInfoStruct.iRestartphase    =  ' ';   /* ignored anyway */
inputInfoStruct.iStatsOpt = SQLU_STATS_NONE; /*don't bother with them*/
inputInfoStruct.iIndexingMode = SQLU_INX_AUTOSELECT;/*let load choose*/
inputInfoStruct.iCpuParallelism  =  0;
inputInfoStruct.iNonrecoverable  =  SQLU_NON_RECOVERABLE_LOAD;
inputInfoStruct.iAccessLevel     =  SQLU_ALLOW_NO_ACCESS;
inputInfoStruct.iLockWithForce   =  SQLU_NO_FORCE;
inputInfoStruct.iCheckPending = SQLU_CHECK_PENDING_CASCADE_DEFERRED;

/* Setup the parameter structure */
paramStruct.piSourceList = &mediaList;
paramStruct.piLobPathList = NULL;
paramStruct.piDataDescriptor = &dataDescriptor;
paramStruct.piActionString = pAction;
paramStruct.piFileType = SQL_DEL;
paramStruct.piFileTypeMod = NULL;
paramStruct.piLocalMsgFileName = localMsgFileName;
paramStruct.piTempFilesPath =   NULL;
paramStruct.piVendorSortWorkPaths = NULL;
paramStruct.piCopyTargetList = NULL;
paramStruct.piNullIndicators = NULL;
paramStruct.piLoadInfoIn = &inputInfoStruct;
paramStruct.poLoadInfoOut    = &outputInfoStruct;
paramStruct.piPartLoadInfoIn  = NULL;
paramStruct.poPartLoadInfoOut = NULL;
paramStruct.iCallerAction = SQLU_INITIAL;

printf("\n  Load table.\n");
printf("    client source file name : %s\n", dataFileName);
printf("    action                  : %s\n", actionString);
printf("    client message file name: %s\n", localMsgFileName);

/* load table */
db2Load (db2Version810,            /* Database version number */
    &paramStruct,            /* In/out parameters      */
    &sqlca);                  /* SQLCA                  */

DB2_API_CHECK("table -- load");

/* free memory allocated */
free(pAction);

/* display load info */
printf("\n  Load info.\n");
printf("    rows read     : %ld\n", outputInfoStruct.oRowsRead);
printf("    rows skipped  : %ld\n", outputInfoStruct.oRowsSkipped);
printf("    rows loaded   : %ld\n", outputInfoStruct.oRowsLoaded);
printf("    rows deleted  : %ld\n", outputInfoStruct.oRowsDeleted);
printf("    rows rejected : %ld\n", outputInfoStruct.oRowsRejected);
printf("    rows committed: %ld\n", outputInfoStruct.oRowsCommitted);

/* display content of the new table */
rc = NewTableDisplay();

/* drop new table */
printf("\n  DROP TABLE newtable\n");
```

```
  EXEC SQL DROP TABLE newtable;
  EMB_SQL_CHECK("new table -- drop");

  return 0;
} /* TbLoad */

int TbLoadQuery(void)
{
  int rc = 0;
  struct sqlca sqlca;
  char tableName[128];
  char loadMsgFileName[128];
  db2LoadQueryStruct loadQueryParameters;
  db2LoadQueryOutputStruct loadQueryOutputStructure;

  printf("\n-----------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  db2LoadQuery -- Load Query\n");
  printf("TO CHECK THE STATUS OF A LOAD OPERATION.\n");

  /* Initialize structures */
  memset(&loadQueryParameters, 0, sizeof(db2LoadQueryStruct));
  memset(&loadQueryOutputStructure, 0, sizeof(db2LoadQueryOutputStruct));

  /* Set up the tablename to query. */
  loadQueryParameters.iStringType = DB2LOADQUERY_TABLENAME;
  loadQueryParameters.piString = tableName;

  /* Specify that we want all LOAD messages to be reported. */
  loadQueryParameters.iShowLoadMessages = DB2LOADQUERY_SHOW_ALL_MSGS;

  /* LOAD summary information goes here. */
  loadQueryParameters.poOutputStruct = &loadQueryOutputStructure;

  /* Set up the local message file. */
  loadQueryParameters.piLocalMessageFile = loadMsgFileName;

  /* call the DB2 API */
  strcpy(tableName, "ORG");
  strcpy(loadMsgFileName, "tbldqry.MSG");

  /* load query */
  db2LoadQuery(db2Version810, &loadQueryParameters, &sqlca);

  if (sqlca.sqlcode == 3523)
  {
    printf("\n  No load of the table '%s' is in progress.\n", tableName);
    printf("  Empty message file '%s' created.\n", loadMsgFileName);
  }
  else
  {
    DB2_API_CHECK("status of load operation -- check");

    printf("\n  Load status has been written to local file %.\n",
           loadMsgFileName);

    printf("    Number of rows read         = %ld\n",
           loadQueryOutputStructure.oRowsRead);

    printf("    Number of rows skipped      = %ld\n",
           loadQueryOutputStructure.oRowsSkipped);

    printf("    Number of rows loaded       = %ld\n",
           loadQueryOutputStructure.oRowsLoaded);

    printf("    Number of rows rejected     = %ld\n",
           loadQueryOutputStructure.oRowsRejected);

    printf("    Number of rows deleted      = %ld\n",
           loadQueryOutputStructure.oRowsDeleted);

    printf("    Number of rows committed    = %ld\n",
           loadQueryOutputStructure.oRowsCommitted);

    printf("    Number of warnings          = %ld\n",
           loadQueryOutputStructure.oWarningCount);
  }

  return 0;
} /* TbLoadQuery */
```

Appendix D. File Formats

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 Integrated 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 Work-sheet 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 beginning the import operation. User-defined distinct types (UDTs) are not made part of the new table column types; instead, the base type is used. Similarly, when exporting to the PC/IXF data file format, UDTs are stored as base data types in the PC/IXF 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.

Related concepts:

- "Table expressions" in the *SQL Reference, Volume 1*

Related reference:

- "Delimited ASCII (DEL) File Format" on page 324
- "Non-delimited ASCII (ASC) File Format" on page 330
- "PC Version of IXF File Format" on page 335
- "Casting between data types" in the *SQL Reference, Volume 1*
- "Assignments and comparisons" in the *SQL Reference, Volume 1*

Related samples:

- "dtformat.out -- HOW TO LOAD AND IMPORT DATA FORMAT EXTENSIONS (C)"
- "dtformat.sqc -- Load and import data format extensions (C)"

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 option on the LOAD command).
- [b] The column delimiter can be specified with the COLDEL option.
- [c] The character string delimiter can be specified with the CHARDEL option.

  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 option.

Example and Data Type Descriptions

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
```

Notes:

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 may 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.

DEL Data Type Descriptions

Table 10. 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.

Table 10. 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**
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.
DECIMAL	A DECIMAL constant with the precision and scale of the field being exported. The DECPLUSBLANK option 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.

Table 10. 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
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 10. 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.

Non-delimited ASCII (ASC) File Format

A non-delimited ASCII (ASC) file is a sequential ASCII file with row delimiters. It 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 may 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 option on the LOAD command). The record delimiter is never interpreted to be part of a field of data.

Example and Data Type Descriptions

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
```

Notes:

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.

ASC Data Type Descriptions

Table 11. 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.
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.

Table 11. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
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.

Table 11. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
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.
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.

Related samples:

- "bldapp -- Builds AIX C application programs"
- "bldsqlj.bat -- Builds a Java embedded SQL (SQLJ) application or applet on Windows"
- "bldsqljs.bat -- Builds a Java embedded SQL (SQLJ) stored procedure on Windows"
- "embprep.bat -- To bind Java embedded SQL (SQLj) programs on Windows"

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 may 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 may 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.

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.

Related reference:

- "Differences Between PC/IXF and Version 0 System/370 IXF" on page 383

PC Version of IXF File Format - Details

PC/IXF Record Types

There are five basic PC/IXF record types:

- header
- table
- column descriptor
- data
- application

and six application subtypes that DB2 UDB 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 may 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 may 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 may 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
   IXFCDLPR       010-BYTE   CHARACTER   datalink properties
   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 may 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.

IXFCDLPR

If the column is a DATALINK column, this field describes the following properties:

- The first character represents the link type, and has a value of `U`.
- The second character represents the link control type. Valid values are `N` for no control, and `F` for file control.
- The third character represents the level of integrity, and has a value of `A` (for database manager controlling all DATALINK values).

- The fourth character represents read permission. Valid values are D for database determined permissions, and F for file system determined permissions.
- The fifth character represents write permission. Valid values are B for blocked access, and F for file system determined permissions.
- The sixth character represents recovery options. Valid values are Y (DB2 will support point-in-time recovery of files referenced in this column), and N (no support).
- The seventh character represents the action that is to be taken when the data file is unlinked. Valid values are R for restore, and D for delete the file.

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 may not 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 may 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
```

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.

Related reference:

- "PC/IXF Data Types" on page 356
- "PC/IXF Data Type Descriptions" on page 363

PC/IXF Data Types

Table 12. 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.

Table 12. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
BLOB_LOCATION_ SPECIFIER, 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.
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*.

Table 12. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
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 12. 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.
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 12. 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.

Table 12. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
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.
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.

Table 12. PC/IXF Data Types (continued)

Name	**IXFCTYPE Value**	**Description**
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 13. 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.		

Related reference:

- “PC/IXF Record Types” on page 337
- “FORCEIN Option” on page 374
- “PC/IXF Data Type Descriptions” on page 363

PC/IXF Data Type Descriptions

Table 14. 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.

Table 14. 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
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.
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.

Table 14. 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
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.
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.

Table 14. 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
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 14. 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.

Table 14. 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
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.
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.

Related reference:

- "PC/IXF Record Types" on page 337
- "PC/IXF Data Types" on page 356

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.
- 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 Table

  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 Table

  Import of a PC/IXF file into an *existing* table is specified by the INSERT, the INSERT_UPDATE, 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: The IMPORT 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: The IMPORT 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.
- Importing or loading PC/IXF files containing DBCS data requires that the corresponding conversion files (located in `sqllib\conv`) be installed on the client machine. The names of these conversion files contain both the source and the target code page numbers; the extension is always `.cnv`. For example, file `09320943.cnv` contains the conversion table for converting code page 932 to 943.

  If the client machine does not have the appropriate conversion files, they can be copied from a server machine to the `sqllib\conv` directory on the client machine. Be sure to copy the files from a compatible platform; for example, if the client is running on a UNIX based operating system, copy the files from a server that is also running on a UNIX based operating system.

Related reference:

- "PC/IXF Data Types" on page 356

- “FORCEIN Option” on page 374

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 may 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 may be lost.

Table 15 summarizes PC/IXF file import into new or existing database tables without the FORCEIN option.

Table 15. Summary of PC/IXF File Import without FORCEIN Option

PC/IXF COLUMN DATA TYPE	DATABASE COLUMN DATA TYPE											
	NUMERIC					CHARACTER			GRAPH	DATETIME		
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[d]	(SBCS, DBCS)[b]	[b]	DATE	TIME	TIME STAMP
Numeric												
-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							
Character												
-(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			
Datetime												
-DATE										N		
										E		
-TIME											N	
											E	
-TIME STAMP												N
												E

Table 15. Summary of PC/IXF File Import without FORCEIN Option (continued)

PC/IXF COLUMN DATA TYPE	DATABASE COLUMN DATA TYPE											
	NUMERIC					CHARACTER			GRAPH	DATETIME		
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[d]	(SBCS, DBCS)[b]	[b]	DATE	TIME	TIME STAMP

Notes:

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.

[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] Data type is not available in DBCS environments.

FORCEIN Option

The FORCEIN option 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.

FORCEIN General Semantics

The following general semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment:

- The FORCEIN option 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 may contain data types or values that cannot be imported without intervention.
- Import with FORCEIN to a *new* table may 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 option, 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.

FORCEIN Code Page Semantics

The following code page semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment:

- The FORCEIN option 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.
- The FORCEIN option 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).
- The FORCEIN option does not imply code page translation.

  Values of data types that are sensitive to the FORCEIN option 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 may be necessary in the case of fixed length target columns.
- When data is imported to an *existing* table using the FORCEIN option:
  - 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 the FORCEIN option 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 the FORCEIN option:
  - 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).

FORCEIN Example

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 table shows:

- 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 invalid or incompatible.

Table 16. Summary of Import Utility Code Page Semantics (New Table). This table assumes there is no conversion table between a and x. If there were, items 3 and 4 would work successfully without the FORCEIN option.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN	
	Without FORCEIN	**With FORCEIN**
SBCS		
(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)
DBCS		
(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)

Table 16. Summary of Import Utility Code Page Semantics (New Table) (continued). This table assumes there is no conversion table between a and x. If there were, items 3 and 4 would work successfully without the FORCEIN option.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	**CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN**	
	Without FORCEIN	**With FORCEIN**

Notes:

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 "FORCEIN Data Type Semantics" on page 380.
6. The `reject` result is a reflection of the rules for invalid or incompatible data types.

The following table shows:

- 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 17. Summary of Import Utility Code Page Semantics (Existing Table). 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
SBCS			
(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
DBCS			
(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
(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

Table 17. Summary of Import Utility Code Page Semantics (Existing Table) (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**
(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

Notes:

1. See the notes for Table 16 on page 377.
2. The `null or reject` result is a reflection of the rules for invalid or incompatible data types.

FORCEIN Data Type Semantics

The FORCEIN option 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 the FORCEIN option in either an SBCS or a DBCS environment (except where noted):

- In SBCS environments, the FORCEIN option 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 option 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 the FORCEIN option.

- In DBCS environments, the FORCEIN option 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 18 summarizes PC/IXF file import into new or existing database tables with the FORCEIN option.

Table 18. Summary of PC/IXF File Import with FORCEIN Option

PC/IXF COLUMN DATA TYPE	**DATABASE COLUMN DATA TYPE**											
	NUMERIC					**CHARACTER**			**GRAPH**	**DATETIME**		
	SMALL INT	**INT**	**BIGINT**	**DEC**	**FLT**	**(0,0)**	**(SBCS, 0)**[e]	**(SBCS, DBCS)**[b]	[b]	**DATE**	**TIME**	**TIME STAMP**
Numeric												
-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							
Character												
-(0,0)						N						
						E	E w/F	E w/F	E w/F	E[c]	E[c]	E[c]

Table 18. Summary of PC/IXF File Import with FORCEIN Option (continued)

PC/IXF COLUMN DATA TYPE	DATABASE COLUMN DATA TYPE											
	NUMERIC					CHARACTER			GRAPH	DATETIME		
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[e]	(SBCS, DBCS)[b]	[b]	DATE	TIME	TIME STAMP
-(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			
Datetime												
-DATE										N		
										E		
-TIME											N	
											E	
-TIME STAMP												N
												E

Note: If a PC/IXF column can be imported into a database column only with the FORCEIN option, 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 option 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.

Related reference:

- "PC/IXF Data Types" on page 356
- "General Rules Governing PC/IXF File Import into Databases" on page 369

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.

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 may occur.

WSF files use a Lotus code point mapping that is not necessarily the same as existing code pages supported by DB2. 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.

Appendix E. Export/Import/Load Utility Unicode Considerations

The export, import, and load utilities are not supported when they are used with a Unicode client connected to a non-Unicode database. Unicode client files are only supported when the Unicode client is connected to a Unicode database.

The DEL, ASC, and PC/IXF file formats are supported for a UCS-2 database, as described in this section. The WSF format is not supported.

When exporting from a UCS-2 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 UCS-2 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. One way to accomplish this is to set DB2CODEPAGE to the code page of the ASCII data file.

The range of valid ASCII delimiters for SBCS and MBCS clients is identical to what is currently supported by DB2® UDB 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 UCS-2 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 UCS-2 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 (unless explicitly specified otherwise using the `codepage` modifier), while the import utility assumes it to be in the code page of the client application.

The `nochecklengths` modifier is always specified for a UCS-2 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.

Restrictions for Code Pages 1394, 1392 and 5488

The import, export and load utilities can now be used to transfer data from the new Chinese code page GB 18030 (code page identifier 1392 and 5488) and the new Japanese code page ShiftJISX 0213 (code page identifier 1394) to DB2 UDB Unicode databases. In addition, the export utility can be used to transfer data from DB2 UDB 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 UDB Unicode database.

Since only connections between a Unicode client and a Unicode server are supported, so 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 may result in expansion. For example, a 2-byte character may 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 UCS-2 database, graphic string data is always in UCS-2 (code page 1200). For applications connected to non-UCS-2 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-UCS-2 database, the graphic string data is in code page 301. For the 932 client applications connected to a UCS-2 database, the graphic string data is in UCS-2.

Appendix F. 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 SQL query specified for the export operation.
db2uimpm.bnd (RR)	Import. Used to insert data from the source data file into the target table.
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 SQL 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 (RR)	Import. Used to perform general initialization processes for an import operation.

Appendix G. Warning, Error and Completion Messages

Messages generated by the various utilities are included among the SQL messages. These messages are generated by the database manager when a warning or error condition has been detected. Each message has a message identifier that consists of a prefix (SQL) and a four- or five-digit message number. There are three message types: notification, warning, and critical. Message identifiers ending with an `N` are error messages. Those ending with a `W` indicate warning or informational messages. Message identifiers ending with a `C` indicate critical system errors.

The message number is also referred to as the *SQLCODE*. The SQLCODE is passed to the application as a positive or negative number, depending on its message type (N, W, or C). N and C yield negative values, whereas W yields a positive value. DB2 returns the SQLCODE to the application, and the application can get the message associated with the SQLCODE. DB2 also returns an *SQLSTATE* value for conditions that could be the result of an SQL statement. Some SQLCODE values have associated SQLSTATE values.

You can use the information contained in this book to identify an error or problem, and to resolve the problem by using the appropriate recovery action. This information can also be used to understand where messages are generated and logged.

SQL messages, and the message text associated with SQLSTATE values, are also accessible from the operating system command line. To access help for these error messages, enter the following at the operating system command prompt:

```
db2 ? SQLnnnnn
```

where *nnnnn* represents the message number. On UNIX based systems, the use of double quotation mark delimiters is recommended; this will avoid problems if there are single character file names in the directory:

```
db2 "? SQLnnnnn"
```

The message identifier accepted as a parameter for the **db2** command is not case sensitive, and the terminating letter is not required. Therefore, the following commands will produce the same result:

```
db2 ? SQL0000N
db2 ? sql0000
db2 ? SQL0000n
```

If the message text is too long for your screen, use the following command (on UNIX based operating systems and others that support the ″more″ pipe):

```
db2 ? SQLnnnnn | more
```

You can also redirect the output to a file which can then be browsed.

Help can also be invoked from interactive input mode. To access this mode, enter the following at the operating system command prompt:

```
db2
```

To get DB2 message help in this mode, type the following at the command prompt (`db2 =>`):

```
? SQLnnnnn
```

The message text associated with SQLSTATEs can be retrieved by issuing:

```
db2 ? nnnnn
 or
db2 ? nn
```

where *nnnnn* is a five-character SQLSTATE value (alphanumeric), and *nn* is a two-digit SQLSTATE class code (the first two digits of the SQLSTATE value).

Appendix H. DB2 Universal Database technical information

Overview of DB2 Universal Database technical information

DB2 Universal Database technical information can be obtained in the following formats:

- Books (PDF and hard-copy formats)
- A topic tree (HTML format)
- Help for DB2 tools (HTML format)
- Sample programs (HTML format)
- Command line help
- Tutorials

This section is an overview of the technical information that is provided and how you can access it.

Categories of DB2 technical information

The DB2 technical information is categorized by the following headings:

- Core DB2 information
- Administration information
- Application development information
- Business intelligence information
- DB2 Connect information
- Getting started information
- Tutorial information
- Optional component information
- Release notes

The following tables describe, for each book in the DB2 library, the information needed to order the hard copy, print or view the PDF, or locate the HTML directory for that book. A full description of each of the books in the DB2 library is available from the IBM Publications Center at www.ibm.com/shop/publications/order

The installation directory for the HTML documentation CD differs for each category of information:

```
htmlcdpath/doc/htmlcd/%L/category
```

where:

- *htmlcdpath* is the directory where the HTML CD is installed.
- *%L* is the language identifier. For example, `en_US`.
- *category* is the category identifier. For example, `core` for the core DB2 information.

In the PDF file name column in the following tables, the character in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d1e80` identifies the English version of the *Administration Guide: Planning* and the file name `db2d1g80` identifies the German version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

Language	**Identifier**
Arabic	w
Brazilian Portuguese	b
Bulgarian	u
Croatian	9
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Romanian	8
Russian	r
Simp. Chinese	c
Slovakian	7
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

No form number indicates that the book is only available online and does not have a printed version.

Core DB2 information

The information in this category cover DB2 topics that are fundamental to all DB2 users. You will find the information in this category useful whether you are a programmer, a database administrator, or you work with DB2 Connect, DB2 Warehouse Manager, or other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/core`.

Table 19. Core DB2 information

Name	Form Number	PDF File Name
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
IBM DB2 Universal Database Glossary	No form number	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Message Reference, Volume 1	GC09-4840	db2m1x80
IBM DB2 Universal Database Message Reference, Volume 2	GC09-4841	db2m2x80
IBM DB2 Universal Database What's New	SC09-4848	db2q0x80

Administration information

The information in this category covers those topics required to effectively design, implement, and maintain DB2 databases, data warehouses, and federated systems.

The installation directory for this category is `doc/htmlcd/%L/admin`.

Table 20. Administration information

Name	Form number	PDF file name
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80

Table 20. Administration information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guide to GUI Tools for Administration and Development	SC09-4851	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Installing and Administering a Satellite Environment	GC09-4823	db2dsx80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Application development information

The information in this category is of special interest to application developers or programmers working with DB2. You will find information about supported languages and compilers, as well as the documentation required to access DB2 using the various supported programming interfaces, such as embedded SQL, ODBC, JDBC, SQLj, and CLI. If you view this information online in HTML you can also access a set of DB2 sample programs in HTML.

The installation directory for this category is `doc/htmlcd/%L/ad`.

Table 21. Application development information

Name	Form number	PDF file name
IBM DB2 Universal Database Application Development Guide: Building and Running Applications	SC09-4825	db2axx80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

Business intelligence information

The information in this category describes how to use components that enhance the data warehousing and analytical capabilities of DB2 Universal Database.

The installation directory for this category is `doc/htmlcd/%L/wareh`.

Table 22. Business intelligence information

Name	Form number	PDF file name
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Installation Guide	GC27-1122	db2idx80

DB2 Connect information

The information in this category describes how to access host or iSeries data using DB2 Connect Enterprise Edition or DB2 Connect Personal Edition.

The installation directory for this category is `doc/htmlcd/%L/conn`.

Table 23. DB2 Connect information

Name	Form number	PDF file name
APPC, CPI-C, and SNA Sense Codes	No form number	db2apx80
IBM Connectivity Supplement	No form number	db2h1x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition	GC09-4833	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Getting started information

The information in this category is useful when you are installing and configuring servers, clients, and other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/start`.

Table 24. Getting started information

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Clients	GC09-4832	db2itx80
IBM DB2 Universal Database Quick Beginnings for DB2 Servers	GC09-4836	db2isx80
IBM DB2 Universal Database Quick Beginnings for DB2 Personal Edition	GC09-4838	db2i1x80
IBM DB2 Universal Database Installation and Configuration Supplement	GC09-4837	db2iyx80
IBM DB2 Universal Database Quick Beginnings for DB2 Data Links Manager	GC09-4829	db2z6x80

Tutorial information

Tutorial information introduces DB2 features and teaches how to perform various tasks.

The installation directory for this category is `doc/htmlcd/%L/tutr`.

Table 25. Tutorial information

Name	Form number	PDF file name
Business Intelligence Tutorial: Introduction to the Data Warehouse	No form number	db2tux80
Business Intelligence Tutorial: Extended Lessons in Data Warehousing	No form number	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	No form number	db2tdx80
Information Catalog Center Tutorial	No form number	db2aix80
Video Central for e-business Tutorial	No form number	db2twx80
Visual Explain Tutorial	No form number	db2tvx80

Optional component information

The information in this category describes how to work with optional DB2 components.

The installation directory for this category is `doc/htmlcd/%L/opt`.

Table 26. Optional component information

Name	Form number	PDF file name
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80

Table 26. Optional component information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6740	N/A

Release notes

The release notes provide additional information specific to your product's release and FixPak level. They also provides summaries of the documentation updates incorporated in each release and FixPak.

Table 27. Release notes

Name	Form number	PDF file name	HTML directory
DB2 Release Notes	See note.	See note.	doc/prodcd/*%L*/db2ir where *%L* is the language identifier.
DB2 Connect Release Notes	See note.	See note.	doc/prodcd/*%L*/db2cr where *%L* is the language identifier.
DB2 Installation Notes	Available on product CD-ROM only.	Available on product CD-ROM only.	

Note: The HTML version of the release notes is available from the Information Center and on the product CD-ROMs. To view the ASCII file:

- On UNIX-based platforms, see the `Release.Notes` file. This file is located in the `DB2DIR/Readme/%L` directory, where `%L` represents the locale name and `DB2DIR` represents:
  – `/usr/opt/db2_08_01` on AIX
  – `/opt/IBM/db2/V8.1` on all other UNIX operating systems
- On other platforms, see the `RELEASE.TXT` file. This file is located in the directory where the product is installed.

Related tasks:

- "Printing DB2 books from PDF files" on page 401

- "Ordering printed DB2 books" on page 402
- "Accessing online help" on page 402
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 406
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 407

Printing DB2 books from PDF files

You can print DB2 books from the PDF files on the *DB2 PDF Documentation* CD. Using Adobe Acrobat Reader, you can print either the entire book or a specific range of pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com

Procedure:

To print a DB2 book from a PDF file:

1. Insert the *DB2 PDF Documentation* CD. On UNIX operating systems, mount the DB2 PDF Documentation CD. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
   - On Windows operating systems:

     `x:\doc\language` directory, where *x* represents the CD-ROM drive letter and *language* represents the two-character territory code that represents your language (for example, EN for English).
   - On UNIX operating systems:

     `/cdrom/doc/%L` directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

Related tasks:

- "Ordering printed DB2 books" on page 402
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 406
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 407

Related reference:

- "Overview of DB2 Universal Database technical information" on page 393

Ordering printed DB2 books

Procedure:

To order printed books:

- Contact your IBM authorized dealer or marketing representative. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/shop/planetwide
- Phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada.
- Visit the IBM Publications Center at www.ibm.com/shop/publications/order

Related tasks:

- "Printing DB2 books from PDF files" on page 401
- "Finding topics by accessing the DB2 Information Center from a browser" on page 404
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 407

Related reference:

- "Overview of DB2 Universal Database technical information" on page 393

Accessing online help

The online help that comes with all DB2 components is available in three types:

- Window and notebook help
- Command line help
- SQL statement help

Window and notebook help explain the tasks that you can perform in a window or notebook and describe the controls. This help has two types:

- Help accessible from the **Help** button
- Infopops

The **Help** button gives you access to overview and prerequisite information. The infopops describe the controls in the window or notebook. Window and notebook help are available from DB2 centers and components that have user interfaces.

Command line help includes Command help and Message help. Command help explains the syntax of commands in the command line processor. Message help describes the cause of an error message and describes any action you should take in response to the error.

SQL statement help includes SQL help and SQLSTATE help. DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the syntax of SQL statements (SQL states and class codes).

Note: SQL help is not available for UNIX operating systems.

Procedure:

To access online help:

- For window and notebook help, click **Help** or click that control, then click **F1**. If the **Automatically display infopops** check box on the **General** page of the **Tool Settings** notebook is selected, you can also see the infopop for a particular control by holding the mouse cursor over the control.
- For command line help, open the command line processor and enter:
  - For Command help:

    ```
    ? command
    ```

    where *command* represents a keyword or the entire command.

  For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.
- For Message help:

  ```
  ? XXXnnnnn
  ```

  where *XXXnnnnn* represents a valid message identifier.

  For example, `? SQL30081` displays help about the SQL30081 message.
- For SQL statement help, open the command line processor and enter:
  - For SQL help:

    ```
    ? 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, while `? 08` displays help for the 08 class code.
  - For SQLSTATE help:

```
help statement
```

where *statement* represents an SQL statement.

For example, `help SELECT` displays help about the SELECT statement.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 404
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 407

Finding topics by accessing the DB2 Information Center from a browser

The DB2 Information Center accessed from a browser enables you to access the information you need to take full advantage of DB2 Universal Database and DB2 Connect. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, metadata, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser is composed of the following major elements:

Navigation tree
The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topics, the glossary, and the master index in the DB2 Information Center.

Navigation toolbar
The navigation toolbar is located in the top right frame of the browser window. The navigation toolbar contains buttons that enable you to search the DB2 Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame
The content frame is located in the bottom right frame of the browser window. The content frame displays topics from the DB2 Information Center when you click on a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index.

Prerequisites:

To access the DB2 Information Center from a browser, you must use one of the following browsers:

- Microsoft Explorer, version 5 or later
- Netscape Navigator, version 6.1 or later

Restrictions:

The DB2 Information Center contains only those sets of topics that you chose to install from the *DB2 HTML Documentation CD*. If your Web browser returns a `File not found` error when you try to follow a link to a topic, you must install one or more additional sets of topics *DB2 HTML Documentation CD*.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon of the category of topics related to your area of interest. A list of subcategories displays underneath the icon.
2. Continue to click the book icons until you find the category containing the topics in which you are interested. Categories that link to topics display the category title as an underscored link when you move the cursor over the category title. The navigation tree identifies topics with a page icon.
3. Click the topic link. The topic displays in the content frame.

To find a topic or term in the master index:

1. In the navigation tree, click the "Index" category. The category expands to display a list of links arranged in alphabetical order in the navigation tree.
2. In the navigation tree, click the link corresponding to the first character of the term relating to the topic in which you are interested. A list of terms with that initial character displays in the content frame. Terms that have multiple index entries are identified by a book icon.
3. Click the book icon corresponding to the term in which you are interested. A list of subterms and topics displays below the term you clicked. Topics are identified by page icons with an underscored title.
4. Click on the title of the topic that meets your needs. The topic displays in the content frame.

Related concepts:

- "Accessibility" on page 413
- "DB2 Information Center for topics" on page 415

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 406
- "Updating the HTML documentation installed on your machine" on page 408
- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 410
- "Searching the DB2 documentation" on page 411

Related reference:

- "Overview of DB2 Universal Database technical information" on page 393

Finding product information by accessing the DB2 Information Center from the administration tools

The DB2 Information Center provides quick access to DB2 product information and is available on all operating systems for which the DB2 administration tools are available.

The DB2 Information Center accessed from the tools provides six types of information.

Tasks Key tasks you can perform using DB2.

Concepts
Key concepts for DB2.

Reference
DB2 reference information, such as keywords, commands, and APIs.

Troubleshooting
Error messages and information to help you with common DB2 problems.

Samples
Links to HTML listings of the sample programs provided with DB2.

Tutorials
Instructional aid designed to help you learn a DB2 feature.

Prerequisites:

Some links in the DB2 Information Center point to Web sites on the Internet. To display the content for these links, you will first have to connect to the Internet.

Procedure:

To find product information by accessing the DB2 Information Center from the tools:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Click the tab of the information type related to the information you are attempting to find.
3. Navigate through the tree and click on the topic in which you are interested. The Information Center will then launch a Web browser to display the information.
4. To find information without browsing the lists, click the **Search** icon to the right of the list.

   Once the Information Center has launched a browser to display the information, you can perform a full-text search by clicking the **Search** icon in the navigation toolbar.

Related concepts:

- “Accessibility” on page 413
- “DB2 Information Center for topics” on page 415

Related tasks:

- “Finding topics by accessing the DB2 Information Center from a browser” on page 404
- “Searching the DB2 documentation” on page 411

Viewing technical documentation online directly from the DB2 HTML Documentation CD

All of the HTML topics that you can install from the *DB2 HTML Documentation CD* can also be read directly from the CD. Therefore, you can view the documentation without having to install it.

Restrictions:

Because the following items are installed from the DB2 product CD and not the *DB2 HTML Documentation CD*, you must install the DB2 product to view these items:

- Tools help
- DB2 Quick Tour
- Release notes

Procedure:

1. Insert the *DB2 HTML Documentation* CD. On UNIX operating systems, mount the *DB2 HTML Documentation CD*. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start your HTML browser and open the appropriate file:
   - For Windows operating systems:

     ```
     e:\Program Files\sqllib\doc\htmlcd\%L\index.htm
     ```

     where *e* represents the CD-ROM drive, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.
   - For UNIX operating systems:

     ```
     /cdrom/Program Files/sqllib/doc/htmlcd/%L/index.htm
     ```

     where */cdrom/* represents where the CD is mounted, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.

Related tasks:

- “Finding topics by accessing the DB2 Information Center from a browser” on page 404
- “Copying files from the DB2 HTML Documentation CD to a Web Server” on page 410

Related reference:

- “Overview of DB2 Universal Database technical information” on page 393

Updating the HTML documentation installed on your machine

It is now possible to update the HTML installed from the *DB2 HTML Documentation CD* when updates are made available from IBM. This can be done in one of two ways:

- Using the Information Center (if you have the DB2 administration GUI tools installed).
- By downloading and applying a DB2 HTML documentation FixPak .

Note: This will NOT update the DB2 code; it will only update the HTML documentation installed from the *DB2 HTML Documentation CD*.

Procedure:

To use the Information Center to update your local documentation:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Ensure your machine has access to the external Internet; the updater will download the latest documentation FixPak from the IBM server if required.
3. Select **Information Center —> Update Local Documentation** from the menu to start the update.
4. Supply your proxy information (if required) to connect to the external Internet.

The Information Center will download and apply the latest documentation FixPak, if one is available.

To manually download and apply the documentation FixPak :

1. Ensure your machine is connected to the Internet.
2. Open the DB2 support page in your Web browser at: www.ibm.com/software/data/db2/udb/winos2unix/support
3. Follow the link for version 8 and look for the ″Documentation FixPaks″ link.
4. Determine if the version of your local documentation is out of date by comparing the documentation FixPak level to the documentation level you have installed. This current documentation on your machine is at the following level: **DB2 v8.1 GA**.
5. If there is a more recent version of the documentation available then download the FixPak applicable to your operating system. There is one FixPak for all Windows platforms, and one FixPak for all UNIX platforms.
6. Apply the FixPak:
   - For Windows operating systems: The documentation FixPak is a self extracting zip file. Place the downloaded documentation FixPak in an empty directory, and run it. It will create a **setup** command which you can run to install the documentation FixPak.
   - For UNIX operating systems: The documentation FixPak is a compressed tar.Z file. Uncompress and untar the file. It will create a directory named `delta_install` with a script called **installdocfix**. Run this script to install the documentation FixPak.

Related tasks:

- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 410

Related reference:

- "Overview of DB2 Universal Database technical information" on page 393

Copying files from the DB2 HTML Documentation CD to a Web Server

The entire DB2 information library is delivered to you on the *DB2 HTML Documentation CD*, so you can install the library on a Web server for easier access. Simply copy to your Web server the documentation for the languages that you want.

Procedure:

To copy files from the *DB2 HTML Documentation CD* to a Web server, use the appropriate path:

- For Windows operating systems:

  ```
  E:\Program Files\sqllib\doc\htmlcd\%L\*.*
  ```

  where *E* represents the CD-ROM drive and *%L* represents the language identifier.
- For UNIX operating systems:

  ```
  /cdrom:Program Files/sqllib/doc/htmlcd/%L/*.*
  ```

  where *cdrom* represents the CD-ROM drive and *%L* represents the language identifier.

Related tasks:

- "Searching the DB2 documentation" on page 411

Related reference:

- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "Overview of DB2 Universal Database technical information" on page 393

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by web browsers. This task describes possible workarounds.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following workaround, especially if you see the following line in the browser Java console:

```
Cannot find class  java/security/InvalidParameterException
```

- On Windows operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `x:Program Files\sqllib\doc\htmlcd\locale\InvalidParameterException.class` file to the `java\classes\java\security\` directory relative to your Netscape browser installation, where *x* represents the CD-ROM drive letter and *locale* represents the name of the desired locale.

  Note: You may have to create the `java\security\` subdirectory structure.
- On UNIX operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `/cdrom/Program Files/sqllib/doc/htmlcd/locale/InvalidParameterException.class` file to the `java/classes/java/security/` directory relative to your Netscape browser installation, where *cdrom* represents the mount point of the CD-ROM and *locale* represents the name of the desired locale.

  Note: You may have to create the `java/security/` subdirectory structure.

If your Netscape browser still fails to display the search input window, try the following:

- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the machine. Then open a new instance of the Netscape browser and try to start the search again.
- Purge the browser's cache.
- Try a different version of Netscape, or a different browser.

Related tasks:

- "Searching the DB2 documentation" on page 411

Searching the DB2 documentation

To search DB2's documentation, you need Netscape 6.1 or higher, or Microsoft's Internet Explorer 5 or higher. Ensure that your browser's Java support is enabled.

A pop-up search window opens when you click the search icon in the navigation toolbar of the Information Center accessed from a browser. If you are using the search for the first time it may take a minute or so to load into the search window.

Restrictions:

The following restrictions apply when you use the documentation search:

- Boolean searches are not supported. The boolean search qualifiers *and* and *or* will be ignored in a search. For example, the following searches would produce the same results:
  - servlets *and* beans
  - servlets *or* beans
- Wildcard searches are not supported. A search on *java** will only look for the literal string *java** and would not, for example, find *javadoc*.

In general, you will get better search results if you search for phrases instead of single words.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

Note: When you perform a search, the first result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in results lists.

Related tasks:

- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 410

Online DB2 troubleshooting information

With the release of DB2® UDB Version 8, there will no longer be a *Troubleshooting Guide*. The troubleshooting information once contained in this guide has been integrated into the DB2 publications. By doing this, we are able to deliver the most up-to-date information possible. To find information on the troubleshooting utilities and functions of DB2, access the DB2 Information Center from any of the tools.

Refer to the DB2 Online Support site if you are experiencing problems and want help finding possible causes and solutions. The support site contains a large, constantly updated database of DB2 publications, TechNotes, APAR (product problem) records, FixPaks, and other resources. You can use the support site to search through this knowledge base and find possible solutions to your problems.

Access the Online Support site at www.ibm.com/software/data/db2/udb/winos2unix/support, or by clicking the **Online Support** button in the DB2 Information Center. Frequently changing information, such as the listing of internal DB2 error codes, is now also available from this site.

Related concepts:

- "DB2 Information Center for topics" on page 415

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 406

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in DB2® Universal Database Version 8:

- DB2 allows you to operate all features using the keyboard instead of the mouse. See "Keyboard Input and Navigation".
- DB2 enables you customize the size and color of your fonts. See "Accessible Display" on page 414.
- DB2 allows you to receive either visual or audio alert cues. See "Alternative Alert Cues" on page 414.
- DB2 supports accessibility applications that use the Java™ Accessibility API. See "Compatibility with Assistive Technologies" on page 414.
- DB2 comes with documentation that is provided in an accessible format. See "Accessible Documentation" on page 414.

Keyboard Input and Navigation

Keyboard Input

You can operate the DB2 Tools using only the keyboard. You can use keys or key combinations to perform most operations that can also be done using a mouse.

Keyboard Focus

In UNIX-based systems, the position of the keyboard focus is highlighted, indicating which area of the window is active and where your keystrokes will have an effect.

Accessible Display

The DB2 Tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font Settings

The DB2 Tools allow you to select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

Non-dependence on Color

You do not need to distinguish between colors in order to use any of the functions in this product.

Alternative Alert Cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with Assistive Technologies

The DB2 Tools interface supports the Java Accessibility API enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible Documentation

Documentation for the DB2 family of products is available in HTML format. This allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

DB2 tutorials

The DB2® tutorials help you learn about various aspects of DB2 Universal Database. The tutorials provide lessons with step-by-step instructions in the areas of developing applications, tuning SQL query performance, working with data warehouses, managing metadata, and developing Web services using DB2.

Before you begin:

Before you can access these tutorials using the links below, you must install the tutorials from the *DB2 HTML Documentation* CD-ROM.

If you do not want to install the tutorials, you can view the HTML versions of the tutorials directly from the *DB2 HTML Documentation CD*. PDF versions of these tutorials are also available on the *DB2 PDF Documentation CD*.

Some tutorial lessons use sample data or code. See each individual tutorial for a description of any prerequisites for its specific tasks.

DB2 Universal Database tutorials:

If you installed the tutorials from the *DB2 HTML Documentation* CD-ROM, you can click on a tutorial title in the following list to view that tutorial.

Business Intelligence Tutorial: Introduction to the Data Warehouse Center
Perform introductory data warehousing tasks using the Data Warehouse Center.

Business Intelligence Tutorial: Extended Lessons in Data Warehousing
Perform advanced data warehousing tasks using the Data Warehouse Center.

Development Center Tutorial for Video Online using Microsoft® Visual Basic
Build various components of an application using the Development Center Add-in for Microsoft Visual Basic.

Information Catalog Center Tutorial
Create and manage an information catalog to locate and use metadata using the Information Catalog Center.

Video Central for e-business Tutorial
Develop and deploy an advanced DB2 Web Services application using WebSphere® products.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 Information Center for topics

The DB2® Information Center gives you access to all of the information you need to take full advantage of DB2 Universal Database™ and DB2 Connect™ in your business. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, the Information Catalog Center, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser has the following features:

Regularly updated documentation
Keep your topics up-to-date by downloading updated HTML.

Search
Search all of the topics installed on your workstation by clicking **Search** in the navigation toolbar.

Integrated navigation tree
Locate any topic in the DB2 library from a single navigation tree. The navigation tree is organized by information type as follows:

- Tasks provide step-by-step instructions on how to complete a goal.
- Concepts provide an overview of a subject.
- Reference topics provide detailed information about a subject, including statement and command syntax, message help, requirements.

Master index
Access the information in topics and tools help from one master index. The index is organized in alphabetical order by index term.

Master glossary
The master glossary defines terms used in the DB2 Information Center. The glossary is organized in alphabetical order by glossary term.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 404
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 406
- "Updating the HTML documentation installed on your machine" on page 408

Appendix I. Notices

IBM may not offer the products, services, or features discussed in this document in all 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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
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 of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation 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.

Other company, product, or service names may be trademarks or service marks of others.

Index

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-4830-00

Spine information:

IBM® DB2 Universal Database™ Data Movement Utilities Version 8