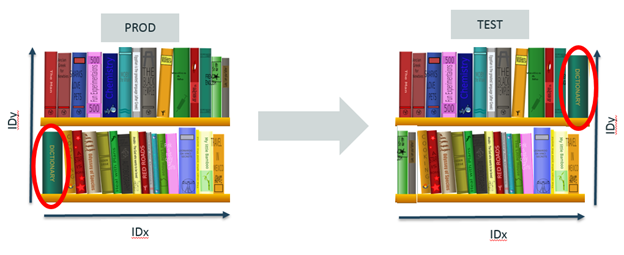
# Data movement options

You can move data within Db2 in several ways: copying a database, copying a Db2 subsystem, or by moving data sets within a particular Db2 subsystem.

**Techniques available to move Db2 data across subsystems**

In large scale organizations, it is often necessary to move data from Db2 Production subsystems to Test subsystems. Of the various technologies that are available to achieve this business objective, circumstances determine which is most appropriate. Additionally, it is sometimes necessary to consider extra technologies such as Encryption or Masking to meet data security expectations.

A picture says a thousand words. The following analogy may help you to digest some concepts that are often difficult for non-Db2 experts. A Db2 subsystem can be represented as a library that stores books (Db2 Tables), and each book contains data. In Db2, each object has an internal identifier, an ID. Those are called OBID “Object ID”. For instance, a Database has an ID, and a table has an ID. Think of a library where books have catalog numbers that are based on their location in the shelves. You can find a book in the shelves from its catalog number; similarly, you can locate a table in Db2 using a couple of IDs.



This concept is important to understand, because Db2 leverages those IDs to make sure that the data is where it belongs. If the IDs do not match, Db2 fails to retrieve the corresponding data. Additionally, it is almost impossible for a Db2 administrator to ensure that a Db2 Object in the Test subsystem has the same object IDs as in the Db2 Production subsystem. As highlighted in the above illustration, the green dictionary (circled in red) is in both the PROD library and the TEST library, but not in the same place on the shelves. That’s typically what PROD and TEST subsystems look like in a Db2 environment.

There are various data movement options available in Db2 like load/unload utility, import utility, export utility, Ingest utility, db2move command etc.

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

|  |  |
| --- | --- |
| Method | Import utility |
| Purpose | To insert data from an external file into a table, hierarchy, view, or nickname |
| Cross platform compatible | Yes |
| Best practice usage | The import utility can be a good alternative to the load utility in the following situations:   * where the target table is a view * the target table has constraints and you don't want the target table to be put in the Set Integrity Pending state * the target table has triggers and you want them fired |
| Import utility | |
| Method | Export utility |
| Purpose | To export data from a database to one of several external file formats. The data can then be imported or loaded at a later time. |
| Cross platform compatible | Yes |
| Best practice usage | This utility is best suited in situations where you want to store data in an external file, to either process it further or move data to another table. High Performance Unload (HPU) is an alternative, however, it must be purchased separately. Export supports XML columns. |
| Export utility | |

# Export/Import/Load utility file formats

Four operating system file formats supported by the Db2 export, import, and load utilities are described:

**DEL**

Delimited ASCII, for data exchange among a wide variety of database managers and file managers. This common approach to storing data uses special character delimiters to separate column values.

**ASC**

Non-delimited ASCII, for importing or loading data from other applications that create flat text files with aligned column data.

**PC/IXF**

PC version of the Integration Exchange Format (IXF), the preferred method for data exchange within the database manager. PC/IXF is a structured description of a database table that contains an external representation of the internal table.

**CURSOR**

A cursor declared against an SQL query. This file type is only supported by the load utility.

When using DEL or ASC data file formats, define the table, including its column names and data types, before importing the file. The data types in the operating system file fields are converted into the corresponding type of data in the database table. The import utility accepts data with minor incompatibility problems, including character data imported with possible padding or truncation, and numeric data imported into different types of numeric fields.

When using the PC/IXF data file format, the table does not need to exist before you begin the import operation. However, the user-defined distinct type (UDT) does need to be defined, otherwise you receive an undefined name error (SQL0204N). Similarly, when you are exporting to the PC/IXF data file format, UDTs are stored in the output file.

When using the CURSOR file type, the table, including its column names and data types, must be defined before beginning the load operation. The column types of the SQL query must be compatible with the corresponding column types in the target table. It is not necessary for the specified cursor to be open before starting the load operation. The load utility will process the entire result of the query associated with the specified cursor whether or not the cursor has been used to fetch rows.

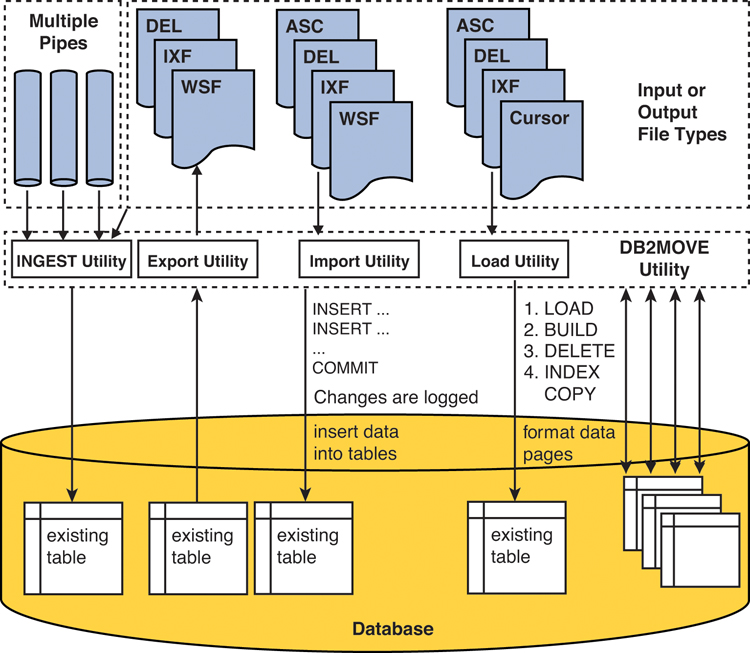
**Maintaining, Backing Up, and Recovering Data**

When working with a database, you insert, update, and delete data constantly. In the long run, you might find the performance of your queries start to degrade as more and more of these operations are performed. This document describes the DB2 utilities you can use to properly maintain your database to avoid these types of problems. In addition, while trying to improve the performance of your system, you might find that it would make more sense to move some data to a reporting system instead of keeping it in your day-to-day operations database, for example. DB2 has several utilities that can help you move data such as EXPORT, IMPORT, LOAD, and so on. Last, there is always some degree of risk when you manipulate important data. To prevent losing any critical information, it is always recommended to back up your databases. DB2 has robust backup and restore capabilities that we will discuss in this document.

### DB2 DATA MOVEMENT UTILITIES: THE BIG PICTURE

Moving data from one database server to another is a common task in a production environment and in almost every phase of the development cycle. For example, a developer might want to export data from a production database and load it into her tables for testing. In a production environment, a database administrator can export a few tables from production to a test database server to investigate a performance problem.

DB2 provides a number of utilities so that you can accomplish these tasks very easily. [Figure 1](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10fig01) presents the big picture of the DB2 data movement utilities. The utilities provide a way to move data from one database to another. The source and target databases can be in the same instance, in different instances on the same server, on different servers on the same platform, or on different platforms entirely. For example, you can move data stored in DB2 on Windows to a database defined in DB2 on a Linux server. Data movement within DB2 is efficient and flexible.



**Figure 1** DB2 data movement utilities

[Figure 1](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10fig01) shows that all data movement utilities use a file either for input or output. The file can be of types DEL, IXF, ASC, WSF, and CURSOR. In the case of the Ingest utility, the input cannot only be a file, but also a named pipe.

To extract data from a table in a database, you use the EXPORT utility. The INGEST, IMPORT, and LOAD utilities insert data from the input files into a specified table. The table must exist prior to using these utilities. A batch version of the data movement utilities, db2move, can export, import, or load multiple tables with just one command. Each utility is discussed in more detail in the sections that follow.

In addition to the utilities illustrated in [Figure 1](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10fig01), there are other tools worth mentioning:

• db2look: This tool is commonly used in conjunction with db2move. It can be used to generate a script file with the structure of database objects or the entire database. Later in the document we provide more details.

• db2relocatedb: Use this tool when you need to rename your database or in situations where you need to move the database from one instance to another in the same server or in different servers. The data is not moved by the tool, but the DB2 internal structures and configuration files are updated appropriately. For more information about this tool, refer to the DB2 Information Center

#### Data Movement File Formats

Before learning about moving data between DB2 databases and/or other data sources, it is important to first understand the file formats that the data movement utilities use. You can choose from five different file formats:

• Delimited ASCII (DEL)

• Non-delimited ASCII (ASC)

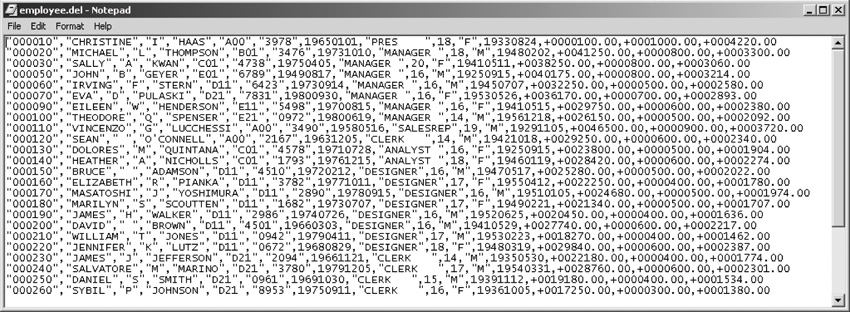
• PC version of Integrated Exchange Format (PC/IXF)

• Worksheet format (WSF)

• CURSOR

##### Delimited ASCII (DEL) Format

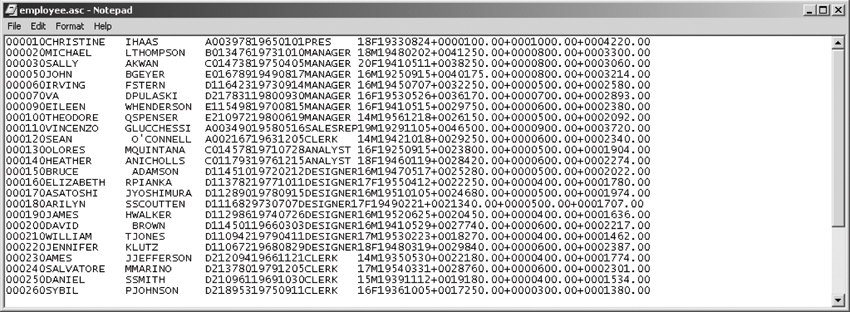
As the name implies, this file format contains a stream of ASCII characters that are separated by row and column delimiters. The comma (,) is the default column delimiter, and the carriage return is the default row delimiter. For character strings, DB2 uses double quotes (" ") as the string delimiter. For example, a DEL file looks similar to [Figure 2](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10fig02). Note that all the string data is surrounded by a pair of double quotes and each column value is separated by a comma.



**Figure 2** Sample DEL file

##### Non-Delimited ASCII (ASC) Format

The ASC file format is also known as **fixed length ASCII file format** because each column length in the file has the same length as defined for the corresponding column definition in the table. For example, variable-length character column definitions in a table are padded with blanks in an ASC file and represented using their maximum length. [Figure 3](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10fig03) shows the same data as in [Figure 2](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10fig02) but in ASC format.



**Figure 3** Sample ASC file

##### PC Version of IXF (PC/IXF) Format

PC/IXF (or simply IXF) files cannot be edited with a normal text editor. They use the IXF data interchange architecture, which is a generic relational database exchange format that lets you move data among DB2 databases. PC/IXF can only be used for moving data between DB2 databases because it is an IBM proprietary format. In addition to data, the file also contains the data types and structure of the table.

**Note**

The PC/IXF format was commonly used with the CREATE and REPLACE\_CREATE options of the IMPORT utility. With these options, the table did not need to exist before running the IMPORT utility because the utility would create the table using the information from the PC/IXF file prior to loading the data. The CREATE and REPLACE\_CREATE options are now deprecated. Using these options to re-create a table did not create the tables with all its required structures.

##### WSF Format

WSF files use worksheet formats that the database manager supports. Any file names with these extensions are accepted: WKS, WK1, WRK, WR1, and WJ2. WSF files are mainly used for moving data between DB2 and these worksheets.

##### CURSOR

You can only use the CURSOR file format with the LOAD utility. The cursor must be declared with an SQL query first before it can be referenced in the load command. For example

DECLARE mycurs CURSOR FOR SELECT col1, col2, col3 FROM test.mytable1  
LOAD FROM mycurs OF CURSOR INSERT INTO test.mytable2

The LOAD utility processes the entire result of the query associated with the specified cursor whether or not the cursor has been used to fetch rows.

#### The DB2 EXPORT Utility

The EXPORT utility extracts data from a table into a file. The export command supports many different options. Let’s start with a simple export command and discuss how to use some options to customize the command. The following example of the export command exports all the rows in the EMPLOYEE table to the file empdata.ixf in IXF format.

export to empdata.ixf of ixf select \* from employee

The following shows the sample output of running this command:

db2 => export to empdata.ixf of ixf select \* from employee  
SQL3104N  The Export utility is beginning to export data to file  
"empdata.ixf".  
SQL3105N  The Export utility has finished exporting "42" rows.  
Number of rows exported: 42

All the keywords in this command are mandatory; that is, you have to provide the output file name, specify the file format, and the SELECT statement that will retrieve the rows to be exported. The exported file can be in a format of DEL, IXF, or WSF. The output file specified is created after executing the export in the directory where the command is executed. Using the optional messages clause, you can specify a file name where warning and error messages of the export operation are logged. If no message file accompanies the messages clause, the messages are written to standard output. Though optional, we highly recommend you use this clause so that all the messages generated by the utility are saved.

The export command also supports SELECT statements with joins, nested statements, and so on. Thus, if you want to export data from two tables, they can be joined as shown in the following example:

export to deptmgr.del of del messages deptmgr.out  
   select deptno, deptname, firstnme, lastname, salary  
     from employee, department  
    where empno = mgrno

The preceding example joins the EMPLOYEE and DEPARTMENT tables to obtain information for each department manager. If the command is successfully executed, the number of rows exported is returned.

Number of rows exported: 8

When the command finishes successfully with no warning or error message, the message file deptmgr.out only includes entries that indicate the beginning and end of the utility execution.

SQL3104N  The Export utility is beginning to export data to file "deptmgr.del".  
  
SQL3105N  The Export utility has finished exporting "8" rows.

##### Specifying Column Names

The method n (column names) option is useful when a column is derived from one or more columns. For example, if you use the following SELECT statement in the export command:

select empno, firstnme, lastname, salary \* 1.3  
  from employee  
where workdept='A00'

The following shows what the output of the SELECT statement would be. Notice that the last column in the select list is a derived column that does not have a column name.

EMPNO  FIRSTNME     LASTNAME        4  
------ ------------ --------------- -------------  
000010 CHRISTINE    HAAS                  130.000  
000110 VINCENZO     LUCCHESSI           60450.000  
000120 SEAN         O'CONNELL           38025.000

The IMPORT utility (which is discussed in a later section) can be executed with a create option that enables you to create the target table if it does not already exist before data is imported. The input file must also contain the definition of the table. If you were to import this result with the create option, the newly created table would have the fourth column named 4. Rather than using a number, you can provide a more descriptive name using the AS clause in the select statement:

export to newsalary.ixf of ixf  
  messages newsalary.out  
  select empno, firstnme, lastname, salary \* 1.3 as new\_salary  
    from employee  
   where workdept='A00'

Alternatively, use the method n option to explicitly specify all the column names. This option is only supported when the export file format is IXF or WSF.

export to newsalary.ixf of ixf  
  messages newsalary.out  
  method n ('EMPLOYEENO', 'FIRSTNAME', 'LASTNAME', 'NEWSALARY')  
  select empno, firstnme, lastname, salary \* 1.3  
    from employee  
   where workdept='A00'

With the method n clause and the specified columns, the resulting file contains the new column names:

EMPLOYEENO FIRSTNAME    LASTNAME        NEWSALARY  
---------- ------------ --------------- -------------  
000010     CHRISTINE    HAAS                  130.000  
000110     VINCENZO     LUCCHESSI           60450.000  
000120     SEAN         O'CONNELL           38025.000

#### The DB2 IMPORT Utility

The IMPORT utility inserts data from an input file into a table or a view. The utility performs inserts as if it were executing INSERT statements. Just like normal insert operations, DB2 validates the data and checks against the table definitions, constraints (such as referential integrity and check constraints), and index definitions. Triggers are also invoked.

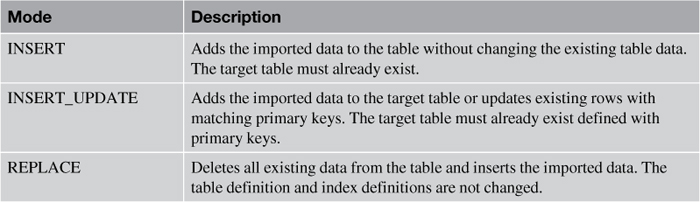
The utility supports options and import modes that enable you to customize its behavior. The import command has many options. Let’s review a simple import command example and discuss the mandatory options. To a certain degree, the import command is structured much like the export command: You have to specify the input file name, format of the file, and the target table name, for example

import from employee.ixf of ixf  
  messages employee.out  
  insert into employee

This command takes the file employee.ixf, which is in the IXF format, as the input and inserts the data rows from the file into the EMPLOYEE table. The IMPORT utility supports input files in ASC, DEL, IXF, and WSF formats. We also recommend that you specify the optional clause messages to save the error and warning messages and the import status. The message file can be used to identify where to restart an interrupted import operation.

##### Import Modes

The previous example uses insert to indicate that new data is to be appended to the existing EMPLOYEE table. [Table 1](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10tab01) lists some of the modes supported by the IMPORT utility.



**Table 1** Import Modes

Let’s take a look at some examples of how to use the import command. In the example that follows, the input data file is employee.del of DEL format. The target table where all the rows will be imported into is the EMPSALARY table; this table must exist to use the replace mode. The warningcount 10 option indicates that the utility will stop after 10 warnings are received. If this option is not specified or is set to zero, the import operation continues regardless of the number of warnings issued.

import from employee.del of del  
  messages empsalary.out  
  warningcount 10  
  replace into empsalary (salary, bonus, comm)

In this next example, the import command deletes all the rows in the table, if table NEWEMPLOYEE exists, and inserts the row contents. If the NEWEMPLOYEE table does not exist, the command creates the table with definitions stored in the IXF input file and inserts the row contents. The number of rows to be imported is limited to the first 1000 rows by using the rowcount 1000 option.

import from employee.ixf of ixf  
  messages employee.out  
  rowcount 1000  
  replace\_create into newemployee

##### Select Columns to Import

There are three ways to select particular columns you want to import. method l uses the starting and ending position (in bytes) for all columns to be imported. This method only supports ASC files; for example

import from employee.asc of asc  
    messages employee.out  
    method l (1 5, 6 14, 24 30)  
    insert into employee

This command imports three selected columns of data into the EMPLOYEE table: bytes 1 to 5 for the first column, bytes 6 to 14 for the second column, and bytes 24 to 30 for the third column.

The other two methods specify the names of the columns (method n) or the field numbers of the input data (method p). method n is only valid for IXF files, and method p can be used with IXF or DEL files. The following examples demonstrate the method n and method p clauses in the import command, respectively.

import from employee.ixf of ixf  
    messages employee.out  
    method n (empno, firstnme, lastname)  
    insert into employee (empno, firstnme, lastname)

import from employee.ixf of ixf  
    messages employee.out  
    method p (1, 2, 4)  
    insert into employee (empno, firstnme, lastname)

##### Regular Commits During an Import

The IMPORT utility inserts data into a table through normal insert operations. Therefore, changes made during the import are logged and are committed to the database upon successful completion of the import operation. By default an import behaves like an atomic compound statement for which more than one insert is grouped into a transaction. If any insert fails, the rest of the inserts are not committed to the database.

If you were to import a few million rows into a table, you would need to make sure there was enough log space to hold the insertions because they are treated as one transaction. However, sometimes it is not feasible to allocate large log space just for the import. The commitcount n option can be used to force a commit after every n records are imported and thus release log space. With commitcount automatic, the utility commits automatically at an appropriate time to avoid running out of active log space and avoid lock escalation.

##### Restarting a Failed Import

If you have import failures due to invalid input, you can, for example, use the message file generated from an import command that uses the commitcount and messages options to identify which record failed. Then you can issue the same import command with restartcount n or skipcount n to start the import from record n+1. Here is an example where skipcount 550 is used so the import restarts from record 551.

import from employee.ixf of ixf  
  commitcount 1000  
  skipcount 550  
  messages newemployee.out  
  create into newemployee in datats index in indexts

#### The DB2 Load Utility

The LOAD utility is another tool you can use to insert data into a table. Note that you cannot run the LOAD utility against a view; the target must be a table that already exists. The major difference between a load and an import is that a load is much faster. Unlike the IMPORT utility, data is not written to the database using normal insert operations. Instead, the LOAD utility reads the input data, formats data pages, and writes directly to the database. Database changes are not logged, and constraint validations (except unique constraint) are not performed during a load operation. Triggers are also not fired.

##### The Load Process

A complete load process consists of four phases:

• Load phase

• Build phase

• Delete phase

• Index copy phase

In the load phase, the LOAD utility scans the input file for any invalid data rows that do not comply with the table definition—for example, if a table column is defined as INTEGER but the input data is stored as "abcd". Invalid data is not loaded into the table. The rejected rows and warnings are written to a dump file specified by the dumpfile modifier. Valid data is then written into the table. At the same time, table statistics (if the statistics use profile option were specified) and index keys are also collected. If the savecount option is specified in the load command, points of consistency are recorded in the message file. Consistency points are established by the LOAD utility; they are useful when it comes to restarting the load operation. You can restart the load from the last successful consistency point.

During the build phase, indexes are produced based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (if the statistics use profile option was specified).

In the load phase, the utility only rejects rows that do not comply with the column definitions. Rows that violated any unique constraint are deleted in the delete phase. Note that only unique constraint violated rows are deleted. Other constraints are not checked during this phase or during any load phase. You have to manually check them after the load operation is complete.

During the index copy phase, index data is copied from a system temporary table space to the original table space. This only occurs if a system temporary table space was specified for index creation during a load operation with the read access option specified.

##### The LOAD Command

The LOAD utility is powerful and has many options. In this book we take a look at the MESSAGES, WARNINGCOUNT, and SAVECOUNT options. Let’s take a look at the example that follows that uses these options.

load from stock.del of del  
  savecount 1000  
  warningcount 10  
  messages stock.out  
  insert into stock(itemid, itemdesc, cost, inventory)

In the example, data in the stock.del delimited input file is loaded into a list of columns of table STOCK. The messages option specifies the filename stock.out to store warnings and errors encountered during the load operation. This particular load stops when the threshold of warnings specified by the warningcount option is encountered—in this example, after 10 warnings. You can check the output file for warnings and errors.

The savecount 1000 option establishes consistency points after every 1,000 rows are loaded. Because a message is issued at each consistency point, ensure that the savecount value is sufficiently high to minimize performance impact.

Consistency points are established during the load phase. You can use them to restart a failed or terminated load operation. By specifying the same load command but replacing insert with the restart option, the load operation automatically continues from the last consistency point.

To terminate a load, issue the same load command but use the terminate option in place of insert; for example

load from stock.del of del  
  savecount 1000  
  warningcount 10  
  messages stock.out  
  terminate

##### Loading from a CURSOR

The LOAD utility supports four file formats: IXF, DEL, ASC, and CURSOR. When using the CURSOR file type as demonstrated in the following example, the cursor must be already declared but does not need to be opened. The entire result of the query associated with the specified cursor will be processed by the LOAD utility. You must also ensure that the column types of the SQL query are compatible with the corresponding column types in the target table.

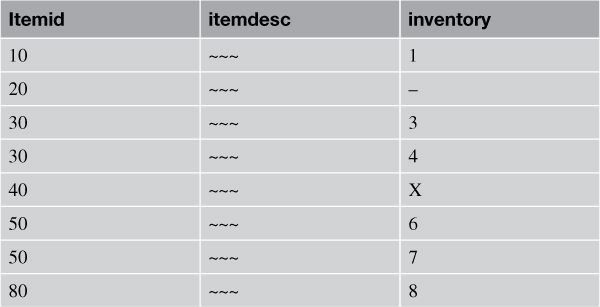
declare cur1 cursor as select \* from oldstock;  
load from cur1 of cursor  
  messages curstock.out  
  insert into stock

##### MODIFIED BY dumpfile and Exception Table

As mentioned earlier, the load process goes through four phases. During the load phase, data that does not comply with the column definition is not loaded. Rejected records can be saved in a dump file by using the modified by dumpfile modifier. If dumpfile is not specified, rejected records are not saved. Because the LOAD utility does not stop unless it reaches the warning threshold if one is specified, it is not easy to identify the rejected records. Hence, it is always a good practice to use the modifier and validate the message file after a load is completed. This example shows how to use modified by dumpfile:

load from stock.ixf of ixf  
  modified by dumpfile=stockdump.dmp  
  messages stock.out  
  replace into stock  
  for exception stockexp

Assume that the input file stock.ixf contains the data shown in [Table 2](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10tab02).



**Table 2** Data Stored in the Input File stock.ixf

The target table STOCK is defined with three columns using this CREATE TABLE statement:

CREATE TABLE stock  
        ( itemid    INTEGER NOT NULL  
        , itemdesc  VARCHAR(100)  
        , inventory INTEGER NOT NULL  
        , PRIMARY KEY (itemid) )

Notice that the second and fifth records in stock.ixf do not comply with the NOT NULL and numeric definitions, respectively. If the load command shown earlier is executed, a dump file (stockdump.dmp) is created to save rows that are not loaded due to incompatible data type and the nullability attribute.  Table 3 shows that the dump file stockdump.dmp contains the rows not loaded.



**Table 3** Rows not Loaded but Stored in the Dump File stockdump.dmp

Recall that in the third load phase, the load process deletes rows that violate any unique constraint defined in the target table. You can save the deleted rows in a table called an **exception table** using the for exception option. If an exception table is not specified, the rows are discarded.

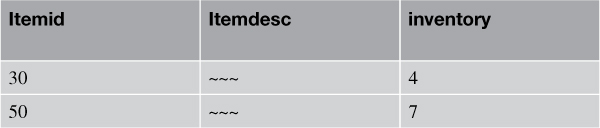
You need to create an exception table manually before you can use it. The table should have the same number of columns, column types, and nullability attributes as the target table to be loaded. You can create such a table with this command:

CREATE TABLE stockexp LIKE stock

Because the LOAD utility does not clean up the exception table, it contains invalid rows from previous load operations unless you remove existing rows before invoking the utility. To log when and why rows are rejected, you can add two other optional columns to the end of the exception table. The first column is defined as a TIMESTAMP data type to record when the record was deleted. The second column is defined as CLOB (32K) or larger and tracks the constraint names that the data violates. To add columns to the table, use the ALTER TABLE statement:

ALTER TABLE stockexp  
  ADD COLUMN load\_ts TIMESTAMP  
  ADD COLUMN load\_msg CLOB(32k)

Like the dumpfile modifier, it is a good practice to also use the exception table, especially if unique violations are possible. The exception table illustrated in [Table 4](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10tab04) contains rows that violated the unique constraints.



**Table 4** Exception Table stockexp

[Figure 4](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10fig04) shows the big picture of the concepts of dumpfile and the exception table.

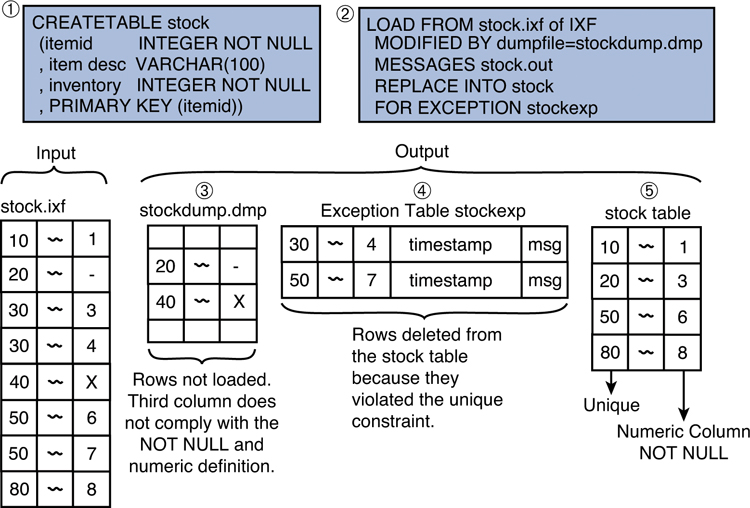
**1.** Create the target table STOCK.

**2.** Issue the load command with modified by dumpfile, messages, and for exception options.

**3.** Rows that do not comply with the table definition (NOT NULL and numeric column) are recorded in the stockdump.dmp file.

**4.** Rows that violated the unique constraint are deleted from the STOCK table and inserted into the exception table.

**5.** Four rows are successfully loaded into the STOCK table.



**Figure 4** Loading data with dumpfile and an exception table

##### Loading from a Client

In all the examples you have seen so far, the load commands are executed from the database server, and the input files are located on the database server. You might sometimes want to invoke a load operation from a remote client as well as using a file that resides at the client. To do so, specify the client keyword in the command as follows:

load client from stock.ixf of ixf  
  modified by dumpfile=stockdump.dmp  
  rowcount 5000  
  messages stock.out  
  tempfiles path c:\loadtemp  
  replace into stock  
  for exception stockexcept  
  lock with force

You cannot load a CURSOR file type from a client. The dumpfile and lobsinfile modifiers (discussed in the following sections) refer to files on the server even when the command includes the client keyword.

**Note**

Use the load client command when the input file resides on the client from which you are issuing the command. Use the dumpfile, tempfile, and lobsinfile modifiers for files located on the DB2 server.

The rowcount option works exactly the same as the one supported by the IMPORT utility. You can control the number of rows to be loaded with this option.

During the load process, the utility uses temporary files. By default, it allocates temporary files on the directory where the load command was issued. The tempfiles option shown in this example is used to explicitly specify a path for this purpose. Notice that the example also uses the replace mode, which replaces the old data in the target table with the new data.

##### Validating Data Against Constraints

The LOAD utility checks for invalid data and unique constraints during the load process. However, other constraints such as referential integrity and check constraints are not validated. DB2 therefore puts target tables defined with these constraints in check pending state, which forces you to manually validate the data before the tables are available for further processing.

The set integrity command gives you the ability to do just that. The command can be as simple as the following example, which immediately validates data against the constraints for table STOCK:

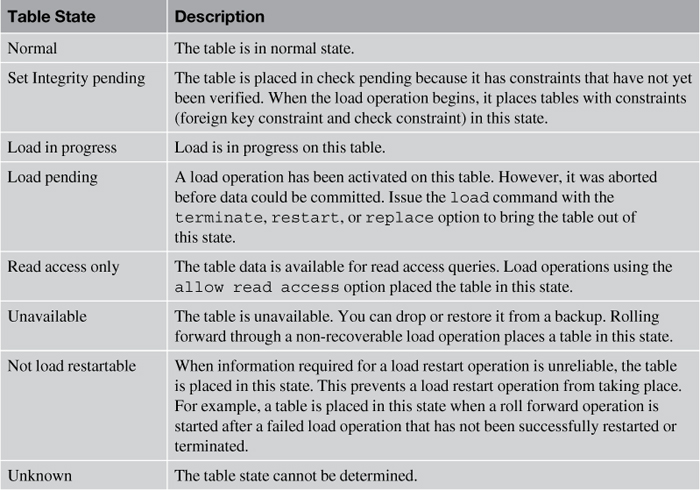
set integrity for stock immediate checked

There are many other options for this command; refer to the DB2 Information Center for more details.

##### Monitoring a Load Operation

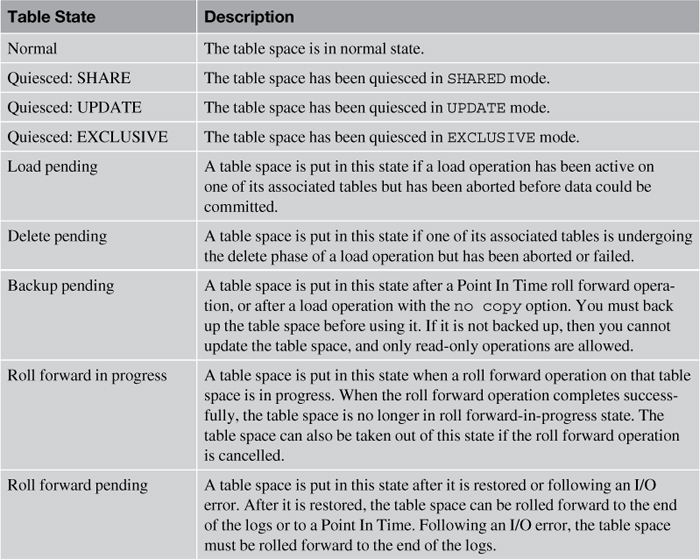
During the phases of a load, the target table and its associated table spaces are in different states. By checking the state of the table and table space, you can tell which phase the load operation is currently in. Before introducing the tools to obtain this information, let’s first discuss the different table and table space states.

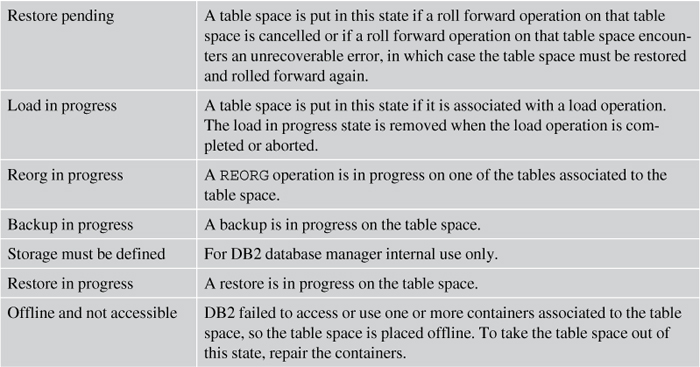
[Table 5](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10tab05) lists the states in which tables can be placed by the database manager. You can control some of these; others are set by the LOAD utility.



**Table 5** Table States

[Table 6](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10tab06) lists the states in which table spaces can be placed by the database manager.





**Table 6** Table Space States

##### Load Querying

DB2 has two utilities that you can use to obtain the table state, load query and list utilities. You can specify the following command to check the status of the load operation on table stock using load query:

load query table stock to c:/stockstatus.out

The output file stockstatus.out might look similar to [Listing 1](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10lis01).

**Listing 1** stockstatus.out Output File

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 "stock.del"  
SQL3500W  The utility is beginning the "LOAD" phase at time "03-21-2013  
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.  
SQL3532I  The Load utility is currently in the "LOAD" phase.  
  
Number of rows read         = 205757  
Number of rows skipped      = 0  
Number of rows loaded       = 205757  
Number of rows rejected     = 0  
Number of rows deleted      = 0  
Number of rows committed    = 123432  
Number of warnings          = 0  
  
Tablestate:  
  Load in Progress

The list utilities command displays the list of active utilities on the instance. Use the show detail option to also display detailed progress information. [Listing 2](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10lis02) illustrates sample output of this utility.

**Listing 2** list utilities Command Output

**list utilities show detail**  
ID                               = 1  
Type                             = LOAD  
Database Name                    = SAMPLE  
Partition Number                 = 0  
Description                      = OFFLINE LOAD Unknown file type AUTOMATIC INDEXING INSERT COPY NO  
Start Time                       = 03/15/2013 00:41:08.767650  
Progress Monitoring:  
   Phase Number                  = 1  
      Description                = SETUP  
      Total Work                 = 0 bytes  
      Completed Work             = 0 bytes  
      Start Time                 = 03/15/2013 00:41:08.786501  
   Phase Number [Current]        = 2  
      Description                = LOAD  
      Total Work                 = 11447 rows  
      Completed Work             = 5481 rows  
      Start Time                 = 03/15/2013 00:41:09.436920

The report in [Listing 2](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10lis02) indicates that a load was performed on the database SAMPLE and includes a brief description of the operation. Progress Monitoring tells you the current phase of the load and the number of rows already loaded and to be loaded.

The table space in which the load target table resides is placed in backup pending state if the COPY NO (the default) option is specified. The utility places the table space in this state at the beginning of the load operation. The table spaces stay in backup pending mode even when the load is complete until you perform a database or table space level backup.

[Listing 3](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/ch10.html#ch10lis03) shows how to retrieve the table space status using the list tablespaces command with the show detail option.

**Listing 3** list tablespaces show detail Command Output

list tablespaces show detail  
  
Tablespace ID                         = 2  
 Name                                 = USERSPACE1  
 Type                                 = System managed space  
 Contents                             = Any data  
  State                                = 0x0000  
   Detailed explanation:  
     Backup pending  
 Total pages                          = 527  
 Useable pages                        = 527  
 Used pages                           = 527  
 Free pages                           = Not applicable  
 High water mark (pages)              = Not applicable  
 Page size (bytes)                    = 4096  
 Extent size (pages)                  = 32  
 Prefetch size (pages)                = 16  
 Number of containers                 = 1

#### The db2move Utility

You can only operate the EXPORT, IMPORT, and LOAD utilities on one table at a time. To move a large number of tables between DB2 databases, use the db2move utility. Supported actions in the command are export, import, load, and copy. Based on the action you request, the utility calls the DB2 export, import, and load application programming interfaces (APIs) accordingly.

The db2move command can also be used without any options. This example exports all tables in the SAMPLE database:

db2move sample export

To import tables with schema dbaadmin and schemas that start with dbauser, you can specify the –tc option and provide a list of schema names; the command also accepts the wildcard (\*).

db2move sample import –tc dbaadmin,dbauser\*

You can also specify the replace mode and lobpath:

db2move sample load –lo replace -l c:\lobpath1,c:\lobpath2

There is no specific authorization prerequisite to invoke this utility. However, the user ID must have the correct authorization and/or privileges for the associated utility (EXPORT, IMPORT, and LOAD) to take action.

The behavior of the export, import, and load actions is exactly the same as described in the previous sections. The only action you probably are not familiar with is copy. It duplicates tables in a schema or schemas into a target database. Only tables with exactly the same schema names specified in the –sn option are copied (via export). If multiple schema names are specified, use commas to separate them without blank spaces. For example

db2move sample copy -sn db2inst1,prodschema  
    -co target\_db acctdb user peter using petepasswd ddl\_and\_load

The db2move command copies supported objects under the schema db2inst1 and prodschema. The –co target\_db option that follows makes the command more interesting. It specifies the target database in which the schemas are going to be copied. This option is mandatory when the copy action is specified. In addition, the target database must be different from the source database. You can provide the user and password with the user and using options when connecting to the target database.

By default, supported objects from the source schema are created, and tables are populated in the target database. This is the behavior of the ddl\_and\_load mode used in the preceding example. Two other modes are available: ddl\_only and load\_only. As the names imply, ddl\_only only creates all the supported objects from the source schema, and load\_only loads all specified tables from the source to the target database. Note that tables must already exist in the target database when this option is used.

Sometimes you might want to rename the schema when copying the objects to the target database. The schema\_map option can be used for this purpose. You simply provide one or more pairs of schema mappings like the following:

schema\_map ((source\_schema1,target\_schema1),(source\_schema2,target\_schema2))

Extra attention is recommended when SCHEMA\_MAP is used. Only the schema of the object itself is renamed; qualified objects inside the object body remain unchanged. For example, in the following view

CREATE VIEW FOO.v1 AS 'SELECT c1 FROM FOO.T1'

if the schema is renamed from FOO to BAR, only the schema of the view is changed. The underlying definition of the view is preserved. This example is mapped to the following. Note that BAR.v1 created in the target database might fail if FOO.T1 is not defined.

CREATE VIEW BAR.v1 AS 'SELECT c1 FROM FOO.T1'

A similar mapping idea also applies to table spaces. For example, you want the copied tables to be stored in a different table space name from the source database. The db2move command is extended to let you specify table space name mappings. Consider the following option:

tablespace\_map ((TS1,TS2),(TS2,TS3), sys\_any)

The preceding table space name mapping indicates that source TS1 is mapped to target TS2, and source TS2 is mapped to target TS3. The sys\_any indicates that the remaining table spaces will use table spaces chosen by the database manager based on the table space selection algorithm. Let’s put the pieces together in an example.

db2move sample copy -sn db2inst1,prodschema  
    -co target\_db acctdb user peter using petepasswd load\_only  
        schema\_map ((db2inst1,db2inst2),(prodschema,devschema))  
        tablespace\_map sys\_any

This command copies supported objects with the schema names db2inst1 and prodschema from the SAMPLE database to the acctdb database. The authorization ID peter and the associated password are used to connect to acctdb. The target tables already exist in acctdb, and the tables are repopulated. All objects under the db2inst1 and prodschema schemas are now under db2inst2 and devschema respectively. Instead of using the table space name defined in the SAMPLE database, the default table space in acctdb is used instead.

### GENERATING DATA DEFINITION LANGUAGE

So far, this document has introduced tools and utilities that you can use to extract data and table definitions using export. In cases when you just want to extract the definition of a table, the db2look command comes in very handy.

db2look extracts the Data Definition Language (DDL) of database objects. Besides that, the tool can also generate the following:

• UPDATE statistics statements

• Authorization statements such as GRANT statements, also known as the Data Control Language (DCL)

• update commands for several Database Manager Configuration parameters and database configuration parameters

• The db2set command for several DB2 registry variables

The following examples demonstrate how the command can be used (the SAMPLE database is being used here):

• This command generates the DDL of objects created by db2admin under the schema prod. It also generates authorization statements. The output file db2look.sql captures this result.

db2look –d sample –u db2admin –z prod –e –x –o db2look.sql

• This command extracts the DDL from the STAFF, DEPARTMENT, and EMPLOYEE tables, and generates UPDATE statements used to replicate statistics of the tables and the associated runstats commands.

db2look –d sample –t staff department employee –m –r

• This command generates the DDL for all the database objects, including the authorization statements, and stores the result in db2look.sql.

db2look –d sample –xd –o db2look.sql

• This command extracts the DDL from the CUSTOMER and PORDER tables. It also exports all files necessary for XML schemas registration. The XSR objects are stored in c:\xsddir.

db2look –d sample –t customer porder –xs –xdir c:\xsddir

In this document, you were introduced to the different data movement utilities that come with DB2. The utilities support the following file formats: DEL, ASC, IXF, WSF, and CURSOR.

The EXPORT utility extracts data from a table or view into a file. The export command can be very simple. At a minimum, you should specify the output file name (where exported data is stored), its file format, the message file name, and a SELECT statement.

The IMPORT utility, on the other hand, inserts data into a specified table or view from a file. You can choose to import to an existing or new table (or view). By default, DB2 only issues one COMMIT at the very end of the import operation. In case of failure during the import, all the changes will be rolled back, and you must restart the import from the beginning. Alternatively, you can use options such as commitcount, restartcount, and skipcount to enable the ability of restarting an import.

The LOAD utility is another method to insert data into a specified table or view and is much faster. The utility formats the data pages while bypassing DB2 buffering and logging. The utility is composed of four phases: load, build, delete, and index copy. You can check the load message file or the status of the table or use the load query command to monitor the load operation.

The db2move utility can be used to move more than one table using just one command. The utility enables you specify the action: export, import, or load. This utility comes in handy when many tables need to be moved.

The RUNSTATS, REORG, and REORGCHK utilities are very important data maintenance utilities that should be performed regularly to ensure that the most optimal data access plans are used. You should also review the automatic database maintenance capabilities DB2 provides. It saves you a lot of time in planning and scheduling maintenance activities.

In this document you also learned about units of work, (UOW), logging methods, and different backup and recovery strategies.

A unit of work (UOW) is composed of one or more statements and completed by a COMMIT or ROLLBACK statement. If a UOW is interrupted in the middle of processing, all statements executed up to that point are rolled back. This ensures data integrity.

When a database is activated, DB2 allocates a number of primary logs based on the setting of the LOGPRIMARY database configuration parameter. When the primary log space fills up, DB2 allocates secondary logs one at a time. When the maximum number of secondary logs is reached, specified by the LOGSECOND database parameter, DB2 encounters a log full condition. At this point, uncommitted transactions are rolled back.

There are three types of database recovery: crash, version, and roll forward recovery. Crash recovery and version recovery are the defaults. To enable roll forward recovery, you need enable archival logging.

There are three logging methods available in DB2: circular, archival, and infinite active logging. Circular logging is the default logging method. However, it does not support roll forward recovery. Archival logging is enabled by setting the LOGARCHMETH1 database configuration parameter. Both archival logging and infinite active logging support roll forward recovery. Log mirroring is valuable in maintaining a redundant active log path.

The RECOVER DATABASE command combines both the RESTORE and ROLLFORWARD commands into one easy step. The RECOVER DATABASE command can continue where it left off if it is interrupted. Alternatively, it is possible to start the recover process from the beginning by specifying the RESTART option.

### CASE STUDY

Assume your company wants to deploy a new accounting application very soon, but the finance department director has demanded more thorough testing. The only test machine available for testing has DB2 for Windows installed. However, you need to obtain data from a DB2 for AIX database server. Because the source and target platforms are different, you cannot restore on Windows a DB2 backup image obtained from DB2 on AIX. In addition, because you don’t need every table and view for your testing, you choose to use data movement utilities to move data to the Windows server.

First, you connect to the source server and then export the required tables with this command:

export to newsalary.ixf of ixf  
  xml to xmlpath  
  xmlfile acctdesc  
  modified by xmlinsepfiles xmlchar  
  messages newsalary.out  
  select empno, firstnme, lastname, salary \* 1.3 as new\_salary  
    from employee  
   where workdept='A00'

You find out that the accounting application needs all of the 100 tables under the schema acct. To save the time and effort of typing the export command for each of the 100 tables, you choose to use the db2move command.

db2move proddb export –sn acct

Because the output files are in IXF format, you can create the tables and import data directly to the target database using the IMPORT utility:

import from newsalary.ixf of ixf  
  xml from xmlpath  
  xmlparse preserve whitespace  
  messages newsalary.out  
  create into newsalary in datats index in indexts

Note that a new table called NEWSALARY is created in the datats table space and that its indexes are stored in the indexts table space.

After the first few successful completions of the import operation, you realize you cannot finish all the imports within the estimated time. The IMPORT utility performs insert statements behind the scenes and thus activates constraint checking, logging, and triggers. The LOAD utility, on the other hand, goes behind the DB2 engine and loads the data directly to the pages. You can choose to perform logging as well as performing only primary and unique key checks. Thus, for the sake of performance, you decide to change the plan and use the LOAD utility instead.

To capture all rows that violated unique constraints of the target table, you create an exception table with this statement:

CREATE TABLE salaryexp  
( empno CHAR(6), firstnme VARCHAR(12), lastname VARCHAR(15)  
, new\_salary DECIMAL(9,2), load\_ts TIMESTAMP, load\_msg CLOB(2K))

You issue the following load command:

load from newsalary.ixf of ixf  
  modified by dumpfile=salarydump.dmp  
  rowcount 5000  
  messages salary.out  
  tempfiles path c:\loadtemp  
  create into salary  
  for exception salaryexp

After the load is completed successfully, the table is not accessible (by default) due to table space backup pending. Therefore, you need to perform a table space or database backup. You choose to perform a simple offline database backup, assuming the database name is targetdb:backup database targetdb

If the table has any constraints defined, such as referential integrity and check constraint, you need to validate the data integrity with the following command:

set integrity for newsalary immediate checked

The target tables should be ready and accessible for testing.

As the testing is progressing, you realize that the performance of the report generator reduces after each monthly inventory update. You first reorganize the tables and update the statistics of the tables with the following commands:

reorg table inventory\_parts indexes all keepdictionary  
  
runstats on table inventory\_parts  
    with distribution default  
    num\_freqvalues 50 num\_quantiles 50  
    util\_impact\_priority 30

Because the table inventory\_parts is enabled for row compression, the KEEPDICTIONARY option is specified to preserve the dictionary. Rather than performing the maintenance tasks manually, you have decided to turn on automatic maintenance. You simply specify the maintenance window and the maintenance tasks you would like to enable, all through IBM Data Studio.

### self assesment QUESTIONS

**1.** Which data movement utility supports the CURSOR input file type?

**2.** What other privileges are needed to load a table if the person already has LOAD authority?

**3.** What will this command do?

db2look –d department –a –e –m –x –f –o db2look.sql

**4.** What is the difference between LOAD and IMPORT utility?

**5.** How you know when to use IMPORT Vs LOAD utility.

**6.** Name and Explain the phases of LOAD process.

**Book Reference:** [DB2 Essentials: Understanding DB2 in a Big Data World, Third Edition](https://learning.oreilly.com/library/view/db2-essentials-understanding/9780133461930/)by Raul F. Chong; Clara LiuPublished by [IBM Press](https://learning.oreilly.com/library/publisher/ibm-press/), 2013