

DB2 Utilities

Presented by DB2 Developer Domain

<http://www7b.software.ibm.com/dmdd/>

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. Introduction	2
2. Data movement utilities	4
3. DB2 EXPORT utility	5
4. DB2 IMPORT utility	10
5. DB2 LOAD utility	13
6. DB2 maintenance utilities	23
7. DB2 Control Center	27
8. DB2 Advisors	29
9. Conclusion	32

Section 1. Introduction

What this tutorial is about

This tutorial introduces utilities available in DB2® to move and maintain data. This is the fifth in a series of six tutorials that you can use to help prepare for the DB2 V8.1 for Linux®, UNIX, and Windows™ Database Administration Certification (Exam 701). The material in this tutorial primarily covers the objectives in Section 5 of the exam, entitled "DB2 Utilities." You can view these objectives at:

<http://www.ibm.com/certify/tests/obj701.shtml>.

You do not need a copy of DB2 Universal Database™ to complete this tutorial. However, if you'd like, you can download a free trial version of *IBM DB2 Universal Database, Enterprise Server Edition*.

In this tutorial, you will learn:

- How to extract data using the EXPORT utility
- How to populate tables with the IMPORT and LOAD utilities
- The advantages and disadvantages of using IMPORT and LOAD
- When and how to use the db2move and db2look commands
- How to use the RUNSTATS, REORG, REORGCHK, and REBIND utilities, and the FLUSH PACKAGE CACHE command
- What can be done with the DB2 Control Center
- When and how to use the DB2 Design and Configuration Advisors

Who should take this tutorial

To take the DB2 UDB V8.1 Database Administration exam, you must have already passed the DB2 UDB V8.1 Family Fundamentals exam (Test 700). You can use the *DB2 Family Fundamentals tutorial series* to prepare for that test. This is a very popular tutorial series that has helped many people to understand the fundamentals of the DB2 family of products.

Although not all materials discussed in the Family Fundamentals tutorial series are required to understand the concepts described in this tutorial, you should at least have a basic knowledge of:

- DB2 instances and databases
- Database objects (table spaces, tables, and indexes)

- Basic SQL operations (UPDATE, INSERT, DELETE, and SELECT statements)
- Different types of constraints
- DB2 Command Line Processor for invoking DB2 commands
- DB2 backup and restore strategies (refer to the *DB2 Backup and Recovery* tutorial for details on this topic, see [Resources](#) on page 32).

This tutorial is one of the tools to help you prepare for Exam 701. You should also review the resources at the end of this tutorial for more information about DB2 utilities (see [Resources](#) on page 32).

About the author

Clara Liu works as a DB2 UDB consultant at the IBM Toronto Laboratory. As a member of the Data Management Channel Development Team, she works closely with IBM business partners and global system integrators. Clara specializes in database application development and integration of new technologies with DB2. She teaches DB2 UDB certification courses to IBM business partners and at conferences. She co-authored the book *DB2 SQL Procedural Language for Windows, UNIX, and Linux* (Prentice Hall, 2002). You can reach Clara at claraliu@ca.ibm.com.

Section 2. Data movement utilities

Utilities and file formats

Three data movement utilities are available in DB2:

- EXPORT
- IMPORT
- LOAD

It is important to make sure that the data you want to transfer is compatible with both the source and target platforms. File formats supported by the utilities are:

- **Non-delimited or fixed-length ASCII (ASC):** As the name implies, this file type contains ASCII data in fixed length to align with column data. 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, which are assumed to be newline characters.
- **Delimited ASCII (DEL):** This is the most common file format used by a variety of database managers for data exchange. It contains ASCII data and uses special character delimiters to separate column values. Rows in the data stream are separated by a newline character as the row delimiter.
- **PC version of the Integrated Exchange Format (PC/IXF):** This is a structured description of a database table. This file format can be used not only to import data but also to create a table that does not already exist in the target database.
- **Worksheet Format (WSF):** Data stored in this format can be interpreted in worksheets. It can be used for export and import only.
- **Cursor:** A cursor is declared with a query. It can only be used as the input of a load operation.

Section 3. DB2 EXPORT utility

Overview of the EXPORT utility

The EXPORT utility extracts data from database tables to a file using an SQL SELECT statement. The exported data can be in the DEL, IXF, or WSF file formats. It is recommended that you include the MESSAGES clause in the export to capture errors, warnings, and informational messages during the export. To successfully invoke the EXPORT utility, you must have SYSADM or DBADM authority, or CONTROL or SELECT privilege on the tables being accessed in the EXPORT command.

Let's look at a simple export example. The command below exports the result of the SELECT statement to a file in DEL format. The message file msg.out records useful information as well as any errors or warnings encountered:

```
EXPORT TO myfile.del OF DEL
  MESSAGES msg.out
  SELECT staff.name, staff.deft, org.location
  FROM org.staff
  WHERE org.deptnum = staff.dept;
```

File type modifiers

In the example on the previous panel, data is extracted to a file in DEL format. By default, column values are separated by commas (,) and character strings are enclosed by double quotation marks ("). What if the data to be extracted already contains commas and double quotes? It will then be impossible for the import or load utility to determine which symbols are actual data and which are delimiters. To customize how EXPORT operates, you can use the MODIFIED BY clause and specify what you want to change with file type modifiers. The EXPORT command will look like this:

```
EXPORT TO file_name OF file_type
  MODIFIED BY file_type_modifiers
  MESSAGES message_file
  select_statement
```

A complete listing of the file type modifiers can be found in the *Command Reference Guide*, under EXPORT. Some commonly used modifiers are listed here for demonstration:

- `chardelx`
 - Specify `x` to be the new single character string delimiter. The default value is a double quotation mark (`"`).
- `coldelx`
 - Specify `x` to be the new single character column delimiter. The default value is a comma (`,`).
- `codepage=x`
 - Specify `x`, an ASCII character string, to be the new code page of the output data. During the export operation, character data is converted to this code page from the application code page.
- `timestampformat="x"`
 - `x` is the format of the time stamp in the source table.

Consider this example:

```
EXPORT TO myfile.del OF DEL
  MODIFIED BY chardel! coldel@ codepage=1208 timestampformat="yyyy.mm.dd hh:mm tt"
  MESSAGES msg.out
  SELECT * FROM schedule
```

The command above exports data from the SCHEDULE table in DEL format with the following behavior:

- Character strings are enclosed by the exclamation mark (`!`).
- Columns are delimited by the `@` sign.
- Character strings are converted to code page 1208.
- The user-defined timestamp in the SCHEDULE table has a format of `yyyy.mm.dd hh:mm tt`.

Exporting large objects

When exporting tables with large object columns, only the first 32 KB of LOB data is exported. This part of the object is placed in the same file as the rest of the column data. To export the LOB data in full and store them in files different from the other column data, you must use the `LOBSINFILE` file modifier. Some other LOB-related options are possible when `LOBSINFILE` is specified. Below is an `EXPORT` command with LOB options:

```
EXPORT TO file_name OF file_type
  LOBS TO lobfile_directory_1, lobfile_directory_2, ...
  LOBFILE lobfilename
  MODIFIED BY LOBSINFILE
```

select_statement

With the `LOBSINFILE` modifier, the `EXPORT` utility looks for the directories specified in the `LOBS TO` clause and places the LOB data there. If no `LOBS TO` clause is found, LOB data is sent to the current working directory. Notice from the command above that you can specify more than one path as the LOB file target directories. There will be at least one file per LOB path, and each file will contain at least one LOB.

It is probably helpful to identify the extracted LOB files with user-specified file names. The `LOBFILE` clause can be used for this purpose. Each LOB file will have a sequence number as the file extension (e.g., `lobfile.001`, `lobfile.002`, `lobfile.003`, etc.).

LOB Location Specifier

When exporting large objects with the `LOBSINFILE` modifier, a LOB Location Specifier (LLS) is generated and stored in the export output file. The LLS is a string used to indicate where LOB data can be found. It has a format of `filename.ext.nnn.mmm/`. Let's look at that in more detail:

- `filename.ext` is the name of the file that contains the LOB data. `ext` is a sequence number, as described in the previous panel.
- `nnn` is the offset of the large object within the LOB file in bytes.
- `mmm` is the length of the large object in bytes.

For example, an LLS of `empresume.001.323.5131/` indicates that the large object is located in the file `empresume.001`, that the actual LOB data begins at an offset of 323 bytes of the file, and that it is 5,131 bytes long.

To clearly illustrate how LLS is used, examine the example below.

```
EXPORT TO empresume.del DEL
  LOBS TO d:\lob1\
  LOBFILE resume MODIFIED BY LOBSINFILE
  SELECT * FROM emp_resume
```

File: empresume.del

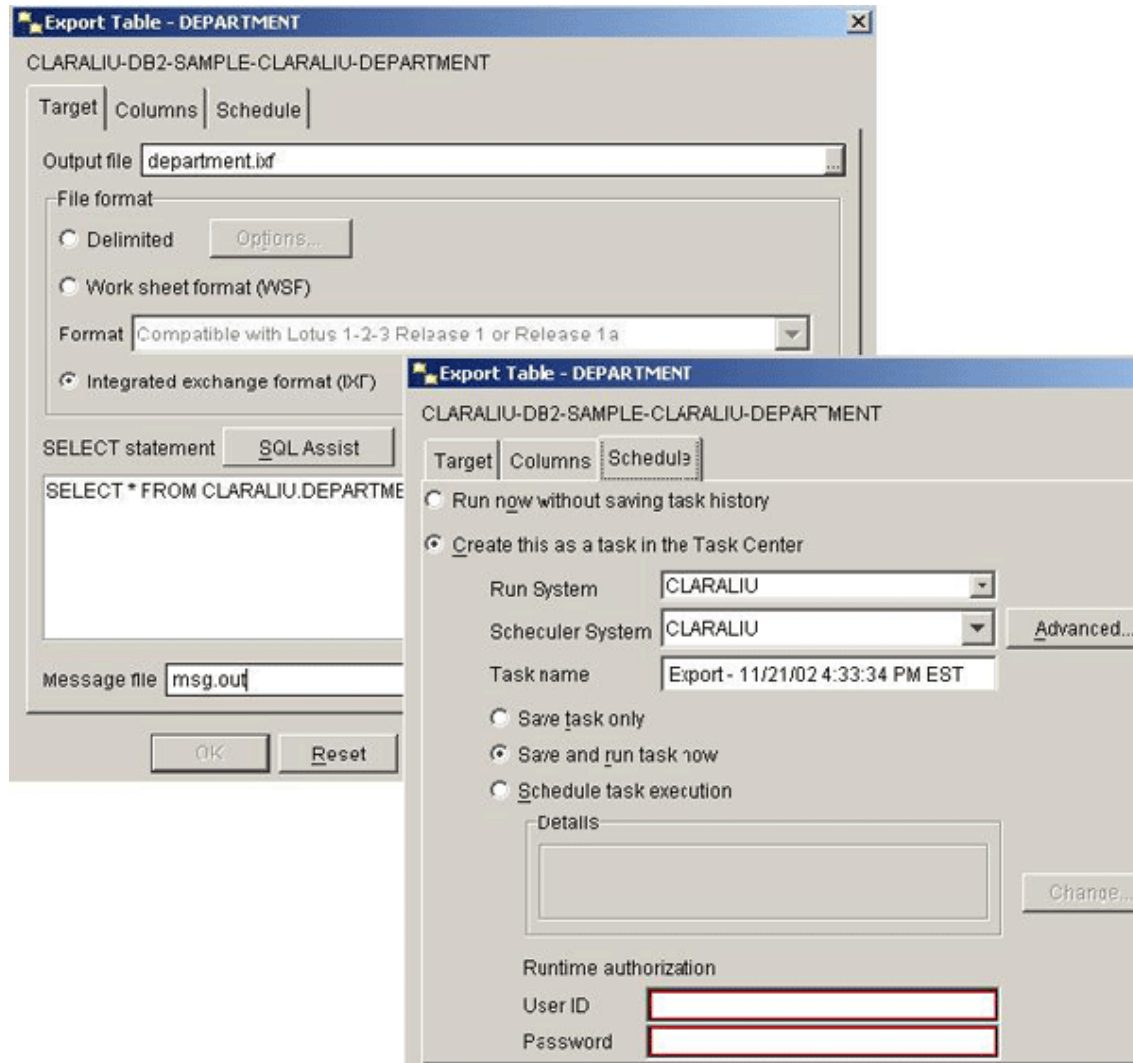
```
"000130", "ascii", "res.001.0.1313/"  
"000130", "script", "res.001.0.1313.1817/"  
"000140", "ascii", "res.001.3130.1316/"  
"000140", "script", "res.001.4446.1878/"  
"000150", "ascii", "res.001.6324.1363/"  
"000150", "script", "res.001.7687.1923/"  
"000190", "ascii", "res.001.9610.1292/"  
"000130", "script", "res.001.10902.1852/"
```

Directory: d:\lob1\

```
res.001  
- contains 8 LOB data
```

Exporting from the Control Center

In addition to executing the `EXPORT` command from the DB2 command line, you can export using the Control Center. From this tool, it is possible to specify all the options and clauses supported by the export. As illustrated in the figure below, the Schedule tab allows you to create a task and schedule the export to run at a given time.



Section 4. DB2 IMPORT utility

Overview of the IMPORT utility

The `IMPORT` utility populates data into a table with an input file in a file type of ASC, DEL, IXF, or WSF. The target can be a table, a typed table, or a view. However, imports to system tables, temporary tables, and materialized query tables are not permitted. It is also recommended that you use the `MESSAGES` clause so that errors, warnings, and informational messages are recorded. To successfully import data, you must have `SYSADM` or `DBADM` authority, or underlying privileges (`SELECT`, `INSERT`, `CONTROL`, or `CREATETAB`) on the target table or database, depending on which option you use. The `IMPORT` command is shown below with five different options:

```
IMPORT FROM file_name OF file_type
      MESSAGES message_file
      [ INSERT | INSERT_UPDATE | REPLACE | REPLACE_CREATE | CREATE ]
      INTO target_table_name
```

- The `INSERT` option inserts imported data to the table. The target table must already exist.
- The `INSERT_UPDATE` inserts data to the table, or updates existing rows of the table with matching primary keys. The target table must exist with a primary key defined.
- The `REPLACE` option deletes all existing data and inserts imported data to an existing target table.
- With the `REPLACE_CREATE` option, if the target table exists, the utility deletes existing data and inserts new data as if the `REPLACE` option were specified. If the target table is not defined, the table and its associated indexes will be created before data is being imported. As you can imagine, the input file must be in PC/IXF format, because that format contains a structured description of an exported table. If the target table is a parent table referenced by a foreign key, `REPLACE_CREATE` cannot be used.
- The `CREATE` option creates the target table and its indexes, then imports data into the new table. The only file format supported is PC/IXF. You can also specify the name of the table space where the new table should be created.

Example:

```
IMPORT FROM emp.ixf OF IXF
      MESSAGES msg.out
      CREATE INTO employee IN datatbsp INDEX IN indtbsp
```

IMPORT options

IMPORT is basically a utility to insert data into a table in bulk. This bulk insert operation is just like a normal insert statement in that the activity is logged, indexes are updated, referential integrity is checked, and table constraints are checked. By default, IMPORT commits only once, at the end of the operation. If a large number of rows are imported or inserted into the table, sufficient transaction logs are required for rollback and recovery. You can request periodic commits to prevent the logs from getting full. By committing the inserts regularly, you also reduce the number of rows being lost if a failure occurs during the import operation. The COMMITCOUNT option forces a COMMIT after a set number of records are imported. Here is an example of how you can use the COMMITCOUNT option:

```
IMPORT FROM myfile.ixf OF IXF
  COMMITCOUNT 500
  MESSAGES msg.out
  INSERT INTO newtable
```

If for some reason the above command fails during its execution, you can use the message file to determine the last row that was successfully imported and committed. Then, you can restart the import with the RESTARTCOUNT option. In the command below, the utility will skip the first 30,000 records before beginning the IMPORT operation.

```
IMPORT FROM myfile.ixf OF IXF
  COMMITCOUNT 500 RESTARTCOUNT 30000
  MESSAGES msg.out
  INSERT INTO newtable
```

File type modifiers

The IMPORT utility also supports file type modifiers to customize the import operation. A complete list of modifiers can be found in the *DB2 Command Reference*, under IMPORT. A few of them are outlined here:

- `compound=x`
 - Uses non-atomic compound SQL to insert data. `x` number of statements will be attempted each time.
- `indexschema=schema`
 - Uses the specified schema for the index during index creation.
- `striptblanks`
 - Truncates any trailing blank spaces when loading data into a variable-length field.

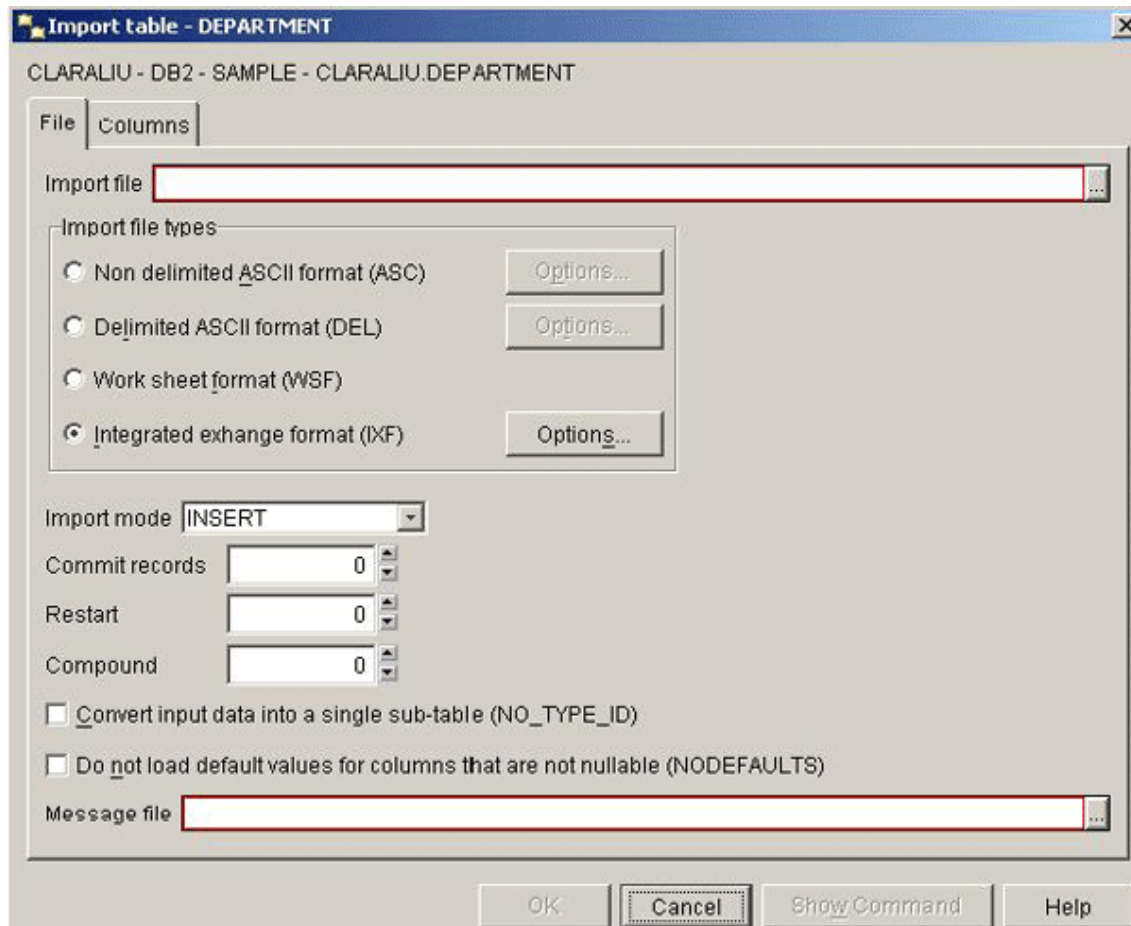
- lobsinfile
 - Indicates that LOB data is being imported. The utility will check the LOBS FROM clause to get the path of the input LOB files.

Here's an example of these file type modifiers in action:

```
IMPORT FOR inputfile.asc OF ASC
  LOBS FROM /u/db2load/lob1, /u/db2load/lob2
  MODIFIED BY compount=5 lobinsfile
  INSERT INTO newtable
```

IMPORT using the Control Center

The Control Center provides easy-to-use graphical interfaces to perform import operations. All the import options and file modifiers discussed in the previous panel are also available in this interface.



Section 5. DB2 LOAD utility

Overview of the LOAD utility

The `LOAD` utility is another method to populate tables with data. Formatted pages are written directly into the database. This mechanism allows more efficient data movement than the `IMPORT` utility. However, some operations, such as referential or table constraints check and triggers invocation, are not performed by the `LOAD` utility.

The following is the core of the `LOAD` command: Other options and modifiers are supported and will be introduced in subsequent panels in this section. To successfully execute this command, you must have `SYSADM`, `DBADM`, or `LOAD` authority, or `INSERT` and/or `DELETE` privileges on the table involved in the load.

```
LOAD FROM input_source OF input_type
      MESSAGES message_file
      [ INSERT | REPLACE | TERMINATE | RESTART ]
      INTO target_tablename
```

The format of a source input for `LOAD` can be `DEL`, `ASC`, `PC/IXF`, or `CURSOR`. A cursor is a result set returned from a `SELECT` statement. An example of using `CURSOR` as the load input is shown here:

```
DECLARE mycursor CURSOR FOR SELECT col1, col2, col3 FROM tabl;
LOAD FROM mycursor OF CURSOR INSERT INTO newtab;
```

The load target must exist before the utility starts. It can be a table, a typed table, or a table alias. Loading to system tables and temporary tables is not supported.

Use the `MESSAGES` option to capture any errors, warnings, and informational messages during the load.

`LOAD` can be executed in four different modes:

- The **INSERT** mode adds input data to a table without changing the existing table data.
- The **REPLACE** mode deletes all existing data from the table and populates it with input data.
- The **TERMINATE** mode terminates a load operation and rolls back to the point in time at which it started. One exception is that if `REPLACE` mode was specified, the table will be truncated.
- The **RESTART** mode is used to restart a previously interrupted load. It will automatically continue from the last consistency point. To use this mode, specify the

same options as in the previous `LOAD` command but with `RESTART`. It allows the utility to find all necessary temporary files generated during the load processing. Therefore, it is important not to manually remove any temporary files generated from a load unless you are sure they are not required. Once the load completes without error, the temporary files will be automatically removed. By default they are created in the current working directory. You can specify the directory where temporary files are stored with the `TEMPFILES PATH` option.

Four phases of a load process

A complete load process has four distinct phases.

1. **Load phase:**

- Loads data into the table.
- Collects index keys and table statistics.
- Records consistency points.
- Places invalid data into dump files and records messages in the message file. When rows of data do not comply with the definition of the table, they are considered to be invalid data and will be rejected (not loaded into the table). Use the `dumpfile` modifier to specify the name and location of a file to record any rejected rows.

2. **Build phase:**

- Creates indexes based on the keys collected during the load phase.

3. **Delete phase:**

- Deletes rows that caused unique key violations and places them in the exception table. Besides data that simply doesn't match the definition of the target table as described above, there may be data that passes the load phase but violates a unique constraint defined in the table. Note that only the unique key-violated rows are considered as bad data here; other constraints are not being checked at this time. Since this type of data is already loaded into the table, the `LOAD` utility will delete the offending rows in this phase. An exception table can be used to store the deleted rows so that you can decide what to do with them after the load operation completes. If no exception table is specified, the offending rows are deleted without a trace. The exception table is discussed in more detail below.
- Records messages in the message file.

4. **Index copy phase:**

- If `ALLOW READ ACCESS` is specified with the `USE TABLESPACE` option, index data is copied from the system temporary table space to the table space where the index should reside.

An *exception table* is a user-defined table that has to have the same column definition of the target table being loaded. If at least one of the columns is not present in the

In the load phase, all the data from the input file is loaded into EMPLOYEE -- except for the two rows marked in pink, as they do not match with the NOT NULL and NUMERIC column definitions. Since the DUMPFILE modifier is specified, these are recorded in the file C:\emp.dmp.

In the delete phase, the two rows marked in yellow are deleted from EMPLOYEE and inserted into the exception table EMPEXP. This is caused by unique violation of the first column in the EMPLOYEE table.

At the end of the load, you should examine the message file, the dump file, and the exception table, then decide how to deal with the rejected rows. If the load completes successfully, the temporary files generated in D:\tmp are removed.

Load options and file type modifiers

Some load options and file type modifiers were introduced in the previous panel. A few more are discussed here.

Load options:

- `ROWCOUNT n`: Allows users to specify only the first `n` records in the input file to be loaded.
- `SAVECOUNT n`: Establishes consistency points after every `n` rows are loaded. Messages are generated and recorded in the message file to indicate how many input rows were successfully loaded at the time of the save point. This point is not possible when input file type is `CURSOR`.
- `WARNINGCOUNT n`: Stops the load after `n` warnings have been raised.
- `INDEXING MODE [REBUILD | INCREMENTAL | AUTOSELECT | DEFERRED]`: In the build phase, indexes are built. This option specifies whether the `LOAD` utility is to rebuild indexes or to extend them incrementally. Four different modes are supported:
 - `REBUILD` mode forces all indexes to be rebuilt.
 - `INCREMENTAL` mode extends indexes with new data only.
 - `AUTOSELECT` mode allows the utility to choose between `REBUILD` and `INCREMENTAL`.
 - `DEFERRED` mode means index create is not going to happen during the load. The indexes involved are marked with refresh required. They will be rebuilt when the database is restated or at the first access to such indexes.
- `STATISTICS [YES | NO]`: After a load is performed, previous statistics of the target table are most likely not valid, as a lot more data has been added. You can choose to collect the statistics by specifying `STATISTICS YES`.

File type modifiers. File type modifiers are specified with the `MODIFIED BY` clause. Here are few you may find useful:

- `fastparse`: Syntax checking on loaded data is reduced to enhance performance.
- `identityignore`, `identitymissing`, and `identityoverride`: Used to ignore, indicate missing, or override identity column data, respectively.
- `indexfreespace n`, `pagefreespace n`, and `totalfreespace n`: Leaves specified amount of free pages in index and data pages.
- `norowwarnings`: Suppresses row warnings.
- `lobsinfile`: Indicates that LOB files are to be loaded; checks `LOBS FROM` option for LOB path.

Table access during load

While a table is being loaded, it is locked by the `LOAD` utility with an exclusive lock. No other access is allowed until the load completes. This is the default behavior of the `ALLOW NO ACCESS` option. During such a load, the table is in the state of `LOAD IN PROGRESS`. There is a handy command that checks the status of a load operation and also returns the table state:

```
LOAD QUERY TABLE table_name
```

You may have guessed that there is an option to allow table access. The `ALLOW READ ACCESS` option causes the table to be locked in share mode. Readers may access the data that already exists in the table but not the new portion. Data that is being loaded is not available until the load is complete. This option puts the loading table in `LOAD IN PROGRESS` state as well as `READ ACCESS ONLY` state.

As mentioned in the last panel, a full index can be rebuilt or an index can be extended with the new data during the build phase. With the `ALLOW READ ACCESS` option, if a full index is being rebuilt, a shadow copy of the index is created. When the `LOAD` utility gets to the index copy phase (see [Four phases of a load process](#) on page 14), the target table is then taken offline and the new index is copied into the target table space.

Regardless of which table access option is specified, various locks are required for the load to process. If the target table is already locked by some application, the `LOAD` utility will have to wait until the locks are released. Instead of waiting for a lock, you can use the `LOCK WITH FORCE` option in the `LOAD` command to force off other applications that hold conflicting locks.

Check pending table state

So far, we know that input data that does not comply with the target table definition is not loaded into the table. Such data is rejected and recorded in the message file at the load phase. In the delete phase, the `LOAD` utility deletes rows that violated any unique constraints. The offending rows are inserted into an exception table if specified. What about other constraints that the table might have defined, such as referential integrity and check constraints? These constraints are not checked by the `LOAD` utility. The table will be placed in `CHECK PENDING` state, which forces you to manually check data integrity before the table can be accessed. Table state can be queried using the `LOAD QUERY` command as discussed in the previous panel. The column `CONST_CHECKED` in the system catalog table `SYSCAT.TABLES` also indicates the status of each constraint defined in the table.

Value	Description
Y	Checked by SYSTEM
N	Not checked (In CHECK PENDING)
U	Checked by USER (As a result of SET INTEGRITY... IMMEDIATE UNCHECKED)
W	Previously checked by USER and some data needs to be verified by SYSTEM (In CHECK PENDING)
F	Byte 5 - materialized query table cannot be refreshed incrementally Byte 7 - content of staging table is incomplete and cannot be used for incremental refresh of the associated materialized query table

```
select tabname, status, CONST_CHECKED from SYSCAT.TABLES
TABNAME      STATUS      CONST_CHECKED
TABLE        N          YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

- Byte 1 - foreign key constraints
- Byte 2 - check constraints
- Byte 5 - materialized query table
- Byte 6 - generated columns
- Byte 7 - staging table

To manually turn off integrity checking for one or more tables, use the `SET INTEGRITY` command. Examples are presented here to demonstrate some options of the command. To check integrity for the appended option of the tables `EMPLOYEE` and `STAFF` immediately, use this command:

```
SET INTEGRITY FOR employee, staff IMMEDIATE CHECKED INCREMENTAL
```

To bypass foreign key checking on table `EMPLOYEE` with the `IMMEDIATE UNCHECKED` option:

```
SET INTEGRITY FOR employee FOREIGN KEY IMMEDIATE UNCHECKED
```

In some cases, you may want to place the target table as well as its descendent tables with foreign key relationship in `CHECK PENDING` state after the load completes. This ensures that all these tables are in control for accessibility until a manual integrity check is performed. The load option is `CHECK PENDING CASCADE IMMEDIATE`, which indicates that the check pending state for foreign key constraints is immediately extended to all descendent foreign key tables. By default, only the loaded table will be placed in check pending state. It is the behavior of load option `CHECK PENDING CASCADE DEFERRED`.

Table space states

Since the `LOAD` utility writes formatted pages into the database directly, no database logging is performed to record the new data being loaded. If you have a recoverable database (i.e., with `LOGREPEAT` and/or `USEREXIT` turned on), DB2 needs to ensure that the database is still recoverable after the load completes. In order to enforce recoverability, the table space where the table is stored will be placed in `BACKUP PENDING` mode. This means that the table space must be backed up before it can be accessed.

This is the default way to make the table space accessible after a load operation completes. Another method is to back up the loaded data while the load is running with the option `COPY YES`. A backup file will be created at the end of the load.

There is another option you may consider if you want to avoid backing up the table space right after the load completes. The load option `NONRECOVERABLE` marks the table being loaded as unrecoverable. The associated table space is fully accessible after load completes. DB2 does not stop you in querying and modifying the table data. However, if later you need to restore the table space and roll forward to a time that passes the `NONRECOVERABLE` load operation, the loaded table is not recoverable. The recovery progress skips all the logs associated with the table. You can only drop and re-create the table. Therefore, it is still recommended that you back up the table space at a convenient time so that the existing and loaded data is saved in a backup.

IMPORT vs. LOAD

Here is a comparison of the IMPORT and LOAD utilities:

Import	Load
Slower on large amounts of data	Faster on large loads - writes formatted pages
Creation of tables & indexes with IXF format	Tables and indexes must exist
WSF supported	WSF not supported
Import into tables and views (Aliases supported)	Load tables only (Aliases supported)
No support for importing into materialized query tables	Support for loading into materialized query tables
All rows logged	Minimal logging performed
Triggers will be fired	Triggers not supported
Temporary space used within the database	Temporary space used outside the database
Constraints validated during import	All unique key is verified during load Other constraints are validated with the SET INTEGRITY command
If interrupted, table is usable with data up to the last commit point	If interrupted, the table is held in LOAD PENDING state. Either restart or restore tables effected.
Run RUNSTATS after import for Statistics	Statistics can be gathered during Load
Import into mainframe database via DB2 Connect	Cannot load into mainframe database
No back-up image required	Backup can be created during load

db2move

db2move is a data movement tool that can be used to move large numbers of tables between DB2 databases. Supported actions in the command are EXPORT, IMPORT, and LOAD. The syntax of db2move can be as simple as:

```
db2move database_name action options
```

A list of user tables is extracted from the system catalog tables, and each table is exported in PC/IXF format. The PC/IXF files can then be imported or loaded into another DB2 database.

Here are some examples. This command imports all tables in the sample database in REPLACE mode with the specified user ID and password:

```
db2move sample import -io replace -u userid -p password
```

And this command loads all tables under the schemas db2admin and db2user in REPLACE mode:

```
db2move sample load -sn db2admin, db2user -lo REPLACE
```

db2look

db2look is a handy tool that can be invoked from the command prompt and the Control Center. The tool can:

- Extract database definition language (DDL) statements from database objects
- Generate UPDATE statements to update database manager and database configuration parameters
- Generate db2set commands to set the DB2 profile registries
- Extract and generate database statistical reports
- Generate UPDATE statements to replicate statistics on database objects

Utilities like LOAD require the existence of a target table. You can use db2look to extract the table's DDL, run it against the target database, and then invoke the load operation. db2look is very easy to use, as the following examples illustrate.

This command generates DDL statements for all objects created by *peter* from the database department, and the output is stored in `alltables.sql`.

```
db2look -d department -u peter -e -o alltables.sql
```

This next command generates:

- DDL for all objects in the database department (specified by options `-d`, `-a`, and `-e`).
- UPDATE statements to replicate the statistics on all tables and indexes in the database (specified by option `-m`).
- GRANT authorization statements (specified by option `-x`).

- UPDATE statements for the database manager and database configuration parameters, and `db2set` commands for the profile registries (specified by option `-f`).

```
db2look -d department -a -e -m -x -f -o db2look.sql
```

Section 6. DB2 maintenance utilities

The RUNSTATS utility

DB2 utilizes a sophisticated cost-based optimizer to determine how data is being accessed. Its decisions are heavily influenced by statistical information about the size of the database tables and indexes. Therefore it is important to keep the database statistics up to date so that an efficient data access plan can be chosen. The `RUNSTATS` utility is used to update statistics about the physical characteristics of a table and the associated indexes. Characteristics include number of records (cardinality), number of pages, average record length, and so on.

Let's use some examples to illustrate the use of this utility. The following command collects statistics on the table `db2user.employee`. Readers and writers are allowed to access the table while the statistics are being calculated:

```
RUNSTATS ON TABLE db2user.employee ALLOW WRITE ACCESS
```

The following command collects statistics on the table `db2user.employee`, as well as on the columns `empid` and `empname` with distribution statistics. While the command is running, the table is only available for read-only requests:

```
RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION ON COLUMNS ( empid, empname )  
ALLOW READ ACCESS
```

The following command collects statistics on table `db2user.employee` and detailed statistics on all its indexes:

```
RUNSTATS ON TABLE db2user.employee AND DETAILED INDEXES ALL
```

The REORG and REORGCHK utilities

Data being added and removed from the database might not be physically placed in a sequential order. In such a case, DB2 must perform additional read operations to access data. This usually means that more disk I/O operations are required, and we all know such operations are costly. In such a case, you should consider physically reorganizing the table to the index so that related data are located close to one other, hence minimizing I/O operations.

`REORG` is a utility to reorganize data for a table and/or index. Although data is physically rearranged, DB2 provides the option of performing this online or offline. Offline `REORG`

by default allows other users to read the table. You may restrict table access by specifying the `ALLOW NO ACCESS` option. Online `REORG` (also called in-place `REORG`) supports no read or write access to the table. Since data pages are rearranged, concurrent applications might have to wait for `REORG` to complete with the current pages. You can easily stop, pause, or resume the process with the appropriate options.

The following examples are pretty self explanatory:

```
REORG TABLE db2user.employee INDEX db2user.idxemp INPLACE ALLOW WRITE ACCESS
REORG TABLE db2user.employee INDEX db2user.idxemp INPLACE PAUSE
```

`REORGCHK` is another data maintenance utility that has an option to retrieve current database statistics or to update the database statistics. It will also generate a report on the statistics with `REORG` indicators. Using the statistics formulae, `REORGCHK` marks the tables or indexes with asterisks (*) if there is a need to `REORG`.

Let's consider some examples. The following command generates a report of the current statistics on all tables that are owned by the runtime authorization ID:

```
REORGCHK CURRENT STATISTICS ON TABLE USER
```

The following command updates the statistics and generates a report on all the tables created under the schema smith:

```
REORGCHK UPDATE STATISTICS ON SCHEMA smith
```

And here's some `REORGCHK` sample output:

Table statistics:

F1: 100 * OVERFLOW / CARD < 5
 F2: 100 * (Effective Space Utilization of Data Pages) > 68
 F3: 100 * (Required Pages / Total Pages) > 80

SCHEMA	NAME	CARD	OV	NP	FP	TSIZE	F1	F2	F3	REORG
SYSIBM	SYSATTRIBUTES	-	-	-	-	-	-	-	-	----
SYSIBM	SYSBUFFERPOOLNODES	-	-	-	-	-	-	-	-	----
SYSIBM	SYSBUFFERPOOLS	1	0	1	1	52	0	-	100	----
SYSIBM	SYSCHECKS	-	-	-	-	-	-	-	-	----
SYSIBM	SYSCODEPROPERTIES	-	-	-	-	-	-	-	-	----
SYSIBM	SYSCOLAUTH	-	-	-	-	-	-	-	-	----

Index statistics:

F4: CLUSTERATIO or normalized CLUSTERFACTOR > 80
 F5: 100*(KEYS*(ISIZE+9)+(CARD-KEYS)*5) / ((NLEAF-NUM_EMPTY_LEAFS)*INDEXPAGESIZE) > 50
 F6: (100-PCTFREE)*((INDEXPAGESIZE-96)/(ISIZE+12))** (NLEVELS-2)*(INDEXPAGESIZE-96) / (KEYS*(ISIZE+9)+(CARD-KEYS)*5) < 100
 F7: 100 * (NUMRIDS DELETED / (NUMRIDS DELETED + CARD)) < 20
 F8: 100 * (NUM EMPTY LEAFS / NLEAF) < 20

SCHEMA	NAME	CARD	LEAF	ELEAF	LVLS	ISIZE	NDEL	KEYS	F4	F5	F6	F7	F8	REORG
Table: SYSIBM.SYSATTRIBUTES														
SYSTEM	IBM33	-	-	-	-	-	-	-	-	-	-	-	-	-----
SYSTEM	IBM34	-	-	-	-	-	-	-	-	-	-	-	-	-----
SYSTEM	IBM35	-	-	-	-	-	-	-	-	-	-	-	-	-----
Table: SYSIBM.SYSBUFFERPOOLNODES														
SYSTEM	IBM69	-	-	-	-	-	-	-	-	-	-	-	-	-----
Table: SYSIBM.SYSBUFFERPOOLS														
SYSTEM	IBM67	1	1	0	1	22	0	1	100	-	-	0	0	-----
SYSTEM	IBM68	1	1	0	1	10	0	1	100	-	-	0	0	-----
Table: SYSIBM.SYSCHECKS														
SYSTEM	IBM37	-	-	-	-	-	-	-	-	-	-	-	-	-----

The REBIND utility and the FLUSH PACKAGE CACHE command

Before a database application program or any SQL statement can be executed, it is precompiled by DB2 and a *package* is produced. A package is a database object that contains compiled SQL statements used in the application source file. DB2 uses the packages to access data referenced in the SQL statements. How does the DB2 optimizer choose the data access plan for these packages? It relies on database statistics at the time the packages are created.

For static SQL statements, packages are created and bound to the database at compile time. If statistics are updated to reflect the physical database characteristics, existing packages should also be updated. The REBIND utility allows you to re-create a package so that the current database statistics can be used. The command is very simple:

```
REBIND PACKAGE package_name
```

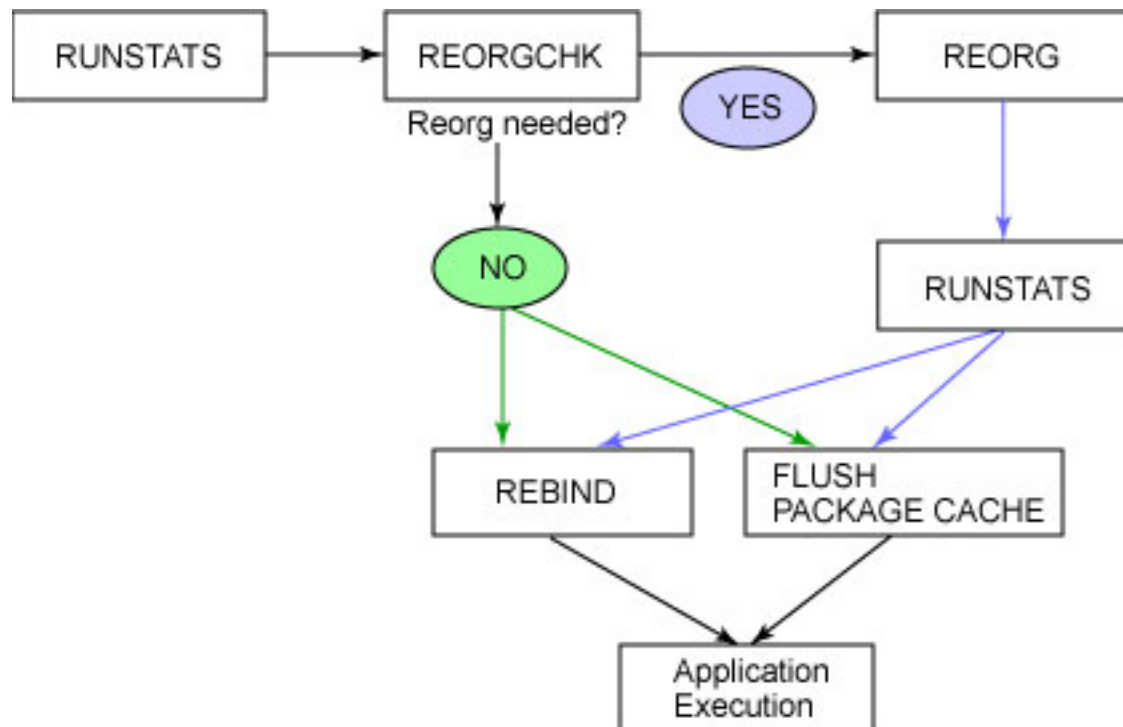
However, if you are going to change the application source, the existing associated package needs to be explicitly dropped and re-created. The REBIND utility is not used for this purpose. We bring this to your attention here because DBAs often misunderstand the use of REBIND.

As for dynamic SQL statements, they are precompiled at run time and stored in the package cache. If statistics are updated, you may flush the cache so that dynamic SQL statements are compiled again to pick up the updated statistics. The command looks like this:

```
FLUSH PACKAGE CACHE DYNAMIC
```

Database maintenance process

Now that you understand RUNSTATS, REORG, REORGCHK, REBIND, and FLUSH PACKAGE CACHE, let's review the data maintenance process that should be performed regularly against your database. The process is illustrated in the following diagram:



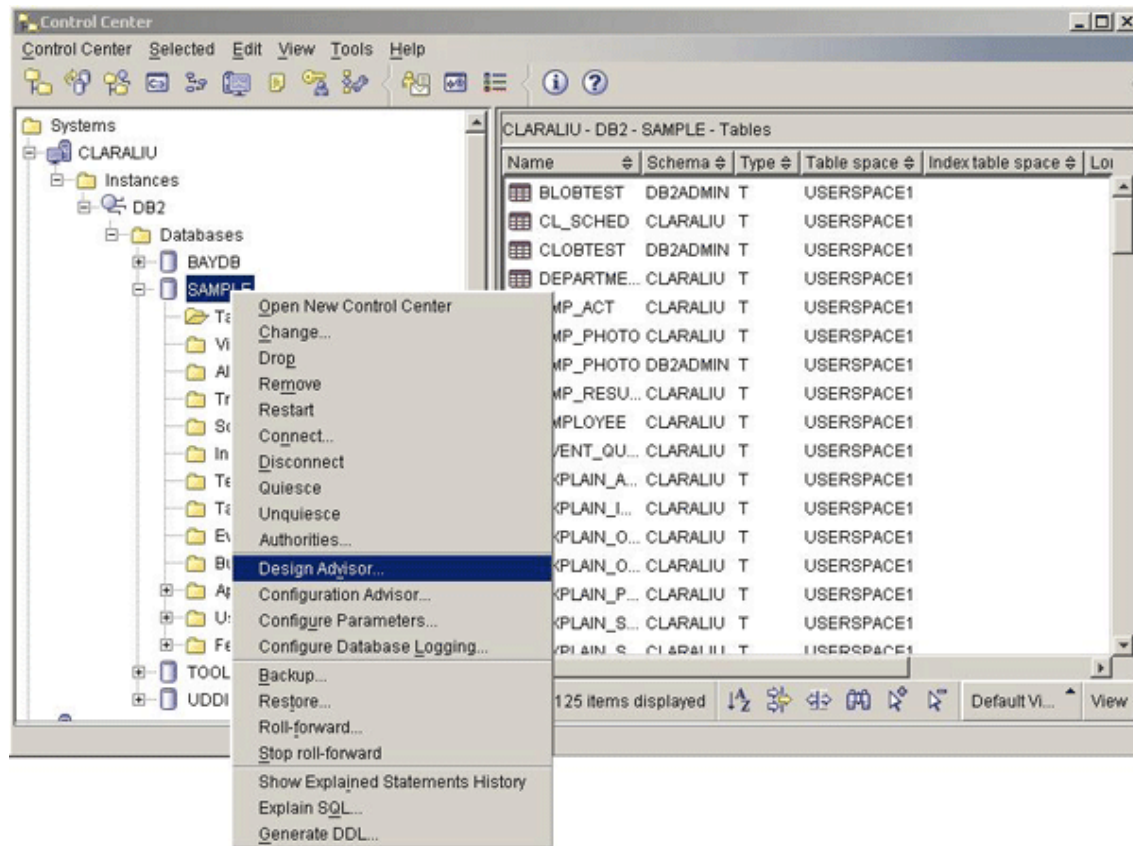
Section 7. DB2 Control Center

DB2 Control Center

By now, you should have used the DB2 Control Center one way or another. It is a graphical administration tool used to manage and administer local or remote DB2 servers. Here are just a few tasks you can perform with the Control Center:

- Configure DB2 instances and databases
- Create databases, table spaces, tables, and indexes using wizards
- Manage DB2 authorities and privileges
- Back up and recover data
- Create tasks and schedule automated jobs
- Export, import, and load data
- Explain problematic SQL statements

These tasks are explained in this tutorial series as well as in the DB2 Fundamentals tutorial series (see [Resources](#) on page 32). It is highly recommended that you use this tool and explore how it can help you manage DB2 servers more efficiently. The Control Center can be conveniently started from the **IBM DB2 => General Administration Tools** program folder from the Start menu on the Windows platforms. You can also start it with the `db2cc` command.



Section 8. DB2 Advisors

DB2 Configuration Advisor

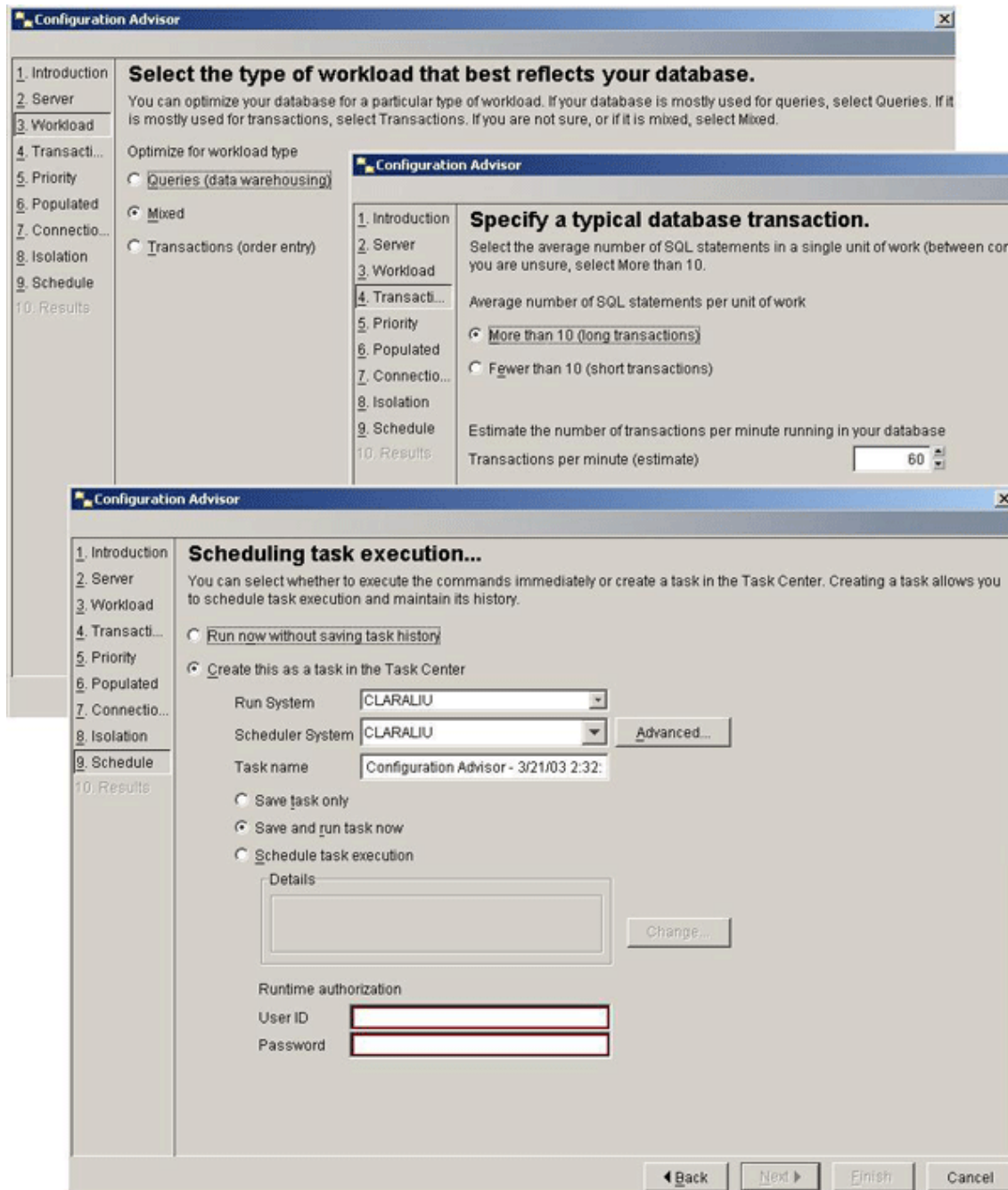
Tuning a database to get optimal performance can be an overwhelming task. There are many different factors that can affect performance. DB2 configuration parameters play an important role in performance because they affect how DB2 behaves. Every application has its own characteristics, and thus requires different DB2 configuration parameter settings.

The DB2 Configuration Advisor can give you advice on reasonable configuration parameter settings based on your responses to the wizard questions. The Advisor prompts you with questions about the hardware configurations, workloads, and database transactions at the installation site. You may save the recommendations in DB2 scripts, review them, and make the changes one at a time.

The Configuration Advisor can be started from the DB2 CLP with the `AUTOCONFIGURE` command. There are approximately 10 input values that you can specify to describe your environment more precisely. Refer to the *DB2 Command Reference* for details. Here is an example of the command:

```
AUTOCONFIGURE USING mem_percent 60 workload_type complex num_stmts 20 APPLY DB AND DBM
```

The Configuration Advisor can also be started from the Control Center, as illustrated in the figure below:



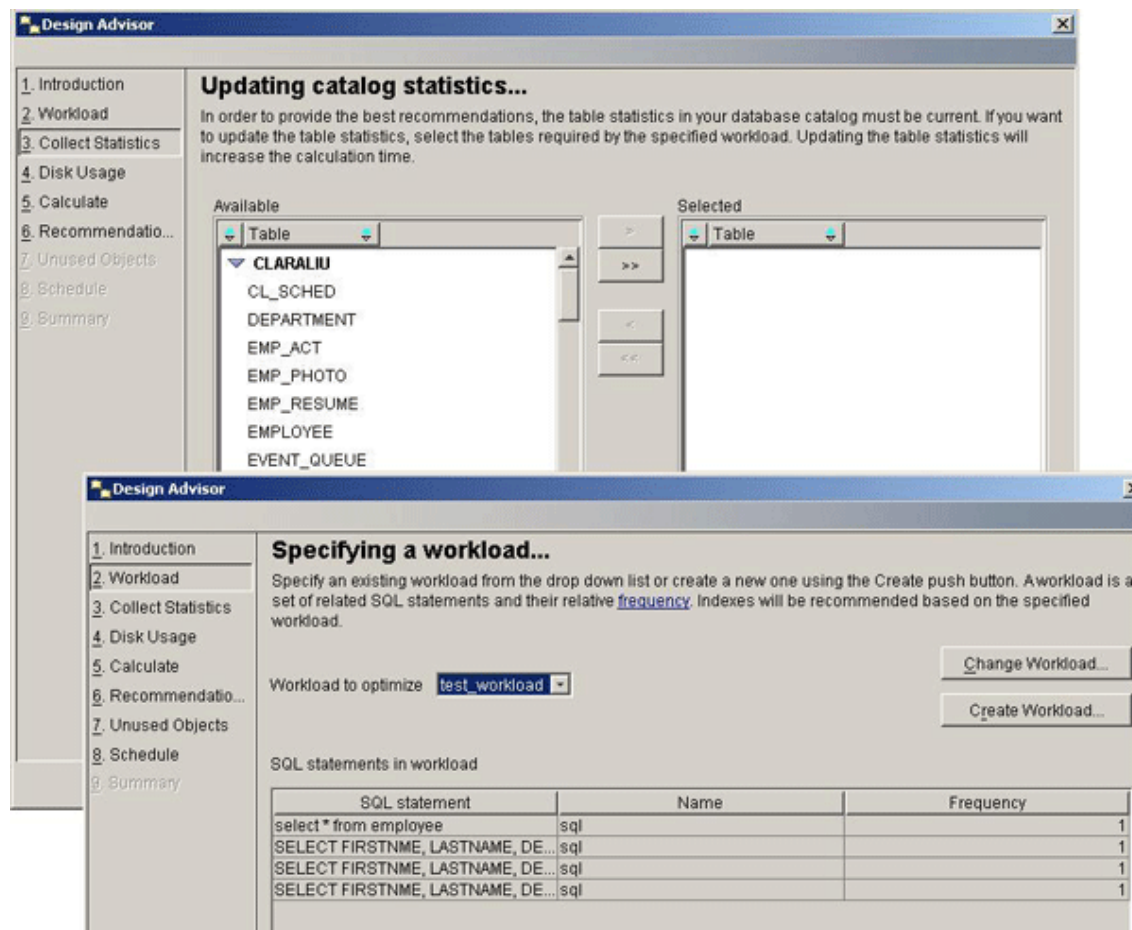
DB2 Design Advisor

The Design Advisor can help you find the best indexes for your SQL statements. It uses the DB2 optimizer, the database statistics, and the Explain mechanism to

generate recommended indexes for a specific query or a set of SQL statements (also called a *workload*). You can start the Design Advisor on the command line with the `db2adv` along with the necessary inputs. The example below executes the Design Advisor against the sample database with an input file `input.sql`, which contains a set of SQL statements. The output is then stored in `output.out`.

```
db2adv -d sample -i input.sql -o output.out
```

The Design Advisor can also be started from the Control Center.



Section 9. Conclusion

Summary

In this tutorial, a number of DB2 utilities were introduced to assist you in maintaining DB2 data. You have learned about:

- Different file formats that DB2 data movement utilities can work with.
- The `EXPORT` utility, which is used to extract data from a table or a view.
- The `IMPORT` utility, which can be used to perform bulk insert into a table or a view.
- The `LOAD` utility, which can also populate table with input data by writing formatted pages into the database directly.
- The four phases of a load operation: load, build, delete, and index copy.
- That a loaded table is placed in the `CHECK PENDING` state if the table has constraints other than unique constraints defined.
- The use of the `SET INTEGRITY` command to remove `CHECK PENDING` state.
- That, for recoverable databases, the table space for which the loaded table is defined will be placed in `BACKUP PENDING` state.
- Different options and file type modifiers to customize the `EXPORT`, `IMPORT`, and `LOAD` operations.
- The purpose of each data maintenance tool: `RUNSTATS`, `REORG`, `REORGCHK`, `REBIND`, and the `FLUSH PACKAGE CACHE` command.
- The use of the Control Center, Design Advisor, and Configuration Advisor.

Good luck with the exam!

Resources

Check out the other parts of the DB2 V8.1 Database Administration certification prep series:

- [Server Management: DB2 V8.1 Database Administration certification prep, Part 1 of 6](#)
- [Data Placement: DB2 V8.1 Database Administration certification prep, Part 2 of 6](#)
- [Database Access: DB2 V8.1 Database Administration certification prep, Part 3 of 6](#)
- [Monitoring DB2 Activity: DB2 V8.1 Database Administration certification prep, Part 4 of 6](#)
- [Backup and Recovery: DB2 V8.1 Database Administration certification prep, Part 6 of 6](#)

You can learn more about DB2 utilities to maintain data from the following resources:

- [DB2 Data Movement Utilities Guide](#)
- [DB2 SQL Reference](#)
- [DB2 Command Reference](#)
- [DB2 Administration Guide - Performance](#)
 - Chapter 5. System Catalog Statistics
 - Chapter 7. SQL Explain Facility - The Design Advisor
 - Chapter 8. Operational Performance - Table management

For more information on the DB2 V8.1 for Linux, UNIX, and Windows Database Administration Certification (Exam 701):

- [IBM Data Management Skills information](#)
- Download a [self-study course for experienced Database Administrators \(DBAs\)](#) to quickly and easily gain skills in DB2 UDB.
- Download a [self-study course for experienced relational database programmers](#) who'd like to know more about DB2.
- [General Certification information](#) -- including some book suggestions, exam objectives, and courses
- For more on DB2 backup and restore strategies, please refer to the [sixth tutorial in this series](#).
- Check out the [IBM Certification Exam Tool \(ICE\)](#), where you can use pre-assessment/sample tests to prepare for exams leading toward IBM certification.
- Check out [developerWorks Toolbox](#) for one-stop access to over 1,000 IBM tools, middleware, and technologies from DB2, Lotus®, Tivoli®, and WebSphere® for open standards-based Web services and application development.

Feedback

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT style sheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial [Building tutorials with the Toot-O-Matic](#) demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.