DB2 9 DBA exam 731 prep, Part 5: DB2 utilities

Get ready for the exam

Skill Level: Introductory

Clara Liu (claraliu@ca.ibm.com)
DB2 UDB Product Management
IBM

30 Jun 2006

Learn skills that help you to properly manage your DB2 database servers. This is the fifth in a series of seven tutorials to help you prepare for the DB2® 9 for Linux®, UNIX®, and Windows™ Database Administration (Exam 731).

Section 1. Before you start

About this series

If you are preparing to take the DB2 DBA certification exam 731, you've come to the right place -- a study hall, of sorts. This series of seven DB2 certification preparation tutorials covers the major concepts you'll need to know for the test. Do your homework here and ease the stress on test day.

About this tutorial

This tutorial introduces skills you must have to properly manage a DB2 server. This is the fifth tutorial in a series of seven tutorials to help you prepare for the DB2 V9 for Linux, UNIX, and Windows; Database Administration Certification (Exam 731).

Objectives

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`, `db2look`, and `db2batch` commands

- How to use the `RUNSTATS`, `REORG`, `REORGCHK`, and `REBIND` utilities, and the `FLUSH PACKAGE CACHE` statement

- What can be done with the DB2 Control Center

- When and how to use the DB2 Design Advisor

Prerequisites

To take the DB2 9 DBA exam, you must have already passed the DB2 9 Fundamentals exam 730. We recommend that you take the DB2 Fundamentals tutorial series before starting this series.

This tutorial is one of the tools to help you prepare for Exam 731. You should also review the resources at the end of this tutorial for more information about DB2 utilities (see Resources).

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

- DB2 products

- DB2 tools

- DB2 instances

- Databases

- Database objects

System requirements

You do not need a copy of DB2 to complete this tutorial. However, you will get more out of the tutorial if you download the free trial version of IBM DB2 9 to work along with this tutorial.

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` or `XQUERY` 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 or views being accessed in the `EXPORT` command.

With the new label-based access control (LBAC) support introduced in DB2 9.1, you need to pay attention to your LBAC credentials, which may or may not allow you to access protected rows and/or columns. When exporting data from a table that has protected rows, your LBAC credentials might limit the rows that are exported. Rows you don't have read access to will not be exported. No error or warning is given. However, if your LBAC credentials does not allow reading from one or more protected columns included in the export, the export will fail with an error.

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.dept, org.location
        FROM org, staff
       WHERE org.deptnumb = 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 with the `MODIFIED BY` clause 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 with the LOBSINFILE modifier

When exporting tables with large object columns, by default 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 LOB options. In DB2 V9.1, you can specify whether you want multiple LOB values to be concatenated and exported in the same output file or each LOB value to be exported to a separate file.

Below is an `EXPORT` command with the `LOBSINFILE` modifier which causes the export utility to write multiple LOB values in the same output file.

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

The `LOBS TO` clause specifies the directories in which the LOB files will be stored. If no `LOBS TO` clause is found, LOB data is sent to the current working directoy. 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.).

When either the `LOBS TO` or `LOBFILE` option is specified, the `LOBSINFILE` behavior will be implicitly activated. However, it is always a good practice to explicitly specify the `LOBSINFILE` modifier to avoid confusion with the `LOBSINSEPFILES` modifier behavior which will be discussed later.

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.lob.nnn.mmm/`. Let's look at that in more detail:

- `filename.ext.lob` 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 `resume.001.lob.1257.2415/` indicates that the large object is located in the file `resume.001.lob`, that the actual LOB data begins at an offset of 1257 bytes of the file, and that it is 2,415 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
    MESSAGES msg.out
    SELECT * FROM emp_resume
```

File: empresume.del

```
"000130","ascii","resume.001.lob.0.1257/","    "
"000130","html","resume.001.lob.1257.2415/","   "
"000140","ascii","resume.001.lob.3672.1261/","  "
"000140","html","resume.001.lob.4933.2440/","   "
"000150","ascii","resume.001.lob.7373.1308/","  "
"000150","html","resume.001.lob.8681.2467/","   "
"000190","ascii","resume.001.lob.11148.1237/",""
"000190","html","resume.001.lob.12385.2402/","  "
```

Directory: d:\lob1

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

Exporting large objects with the LOBSINSEPFILES modifier

As mentioned in the previous sections, you can also choose to export LOB data in full and store each of them in separate files. The LOB options described earlier remain the same, except that the LOBSINSEPFILES modifier is used instead.

Here is an example with such modifier.

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

With this `EXPORT` command, the export utility will write LOB data in files with names resume.ext.lob (i.e. resume.001.lob, resume.002.lob, resume.003.lob, etc.) which are all located in the LOB path d:\lob1.

Exporting XML data

With the introduction of native XML support in DB2 9.1, the export utility is extended to also support XML. If you were to export a table (defined with XML data) without specifying any XML related options, the associated XML data will be written to a file or files separate from the rest of the exported relational data. Let's look at an example. The following `EXPORT` command is issued on the PRODUCT table which has one XML column defined:

```
EXPORT TO prodexport.del DEL
    MESSAGES msg.out
    SELECT * FROM product
```

In this example, the export utility will generate two output files. One of them is the prodexport.del file which contains relational data of the table as well as the XML data specifier (XDS).

File: prodexport.del

```
"100-100-01",,,,,,,"<XDS FIL='prodexport.del.001.xml' OFF='0' LEN='252' />"
"100-101-01",,,,,,,"<XDS FIL='prodexport.del.001.xml' OFF='252' LEN='271' />"
"100-103-01",,,,,,,"<XDS FIL='prodexport.del.001.xml' OFF='523' LEN='304' />"
"100-201-01",,,,,,,"<XDS FIL='prodexport.del.001.xml' OFF='827' LEN='222' />"
```

XDS is a string represented as an XML tag named "XDS". It has attributes that describe information about the actual XML data in the column. Here are the attributes you might see in an XDS string.

- `FIL` specifies the name of the file that contains the XML data.

- `OFF` specifies the byte offset of the XML data in the file named by the FIL attribute.

- `LEN` specifies the length in bytes of the XML data in the file named by the FIL attribute.

- `SCH` specifies the fully qualified SQL identifier of the XML schema used to validate this XML document. This attribute will be discussed in the next panel.

From the content of prodexport.del above, you can see that the first XML data is stored in prodexport.del.001.xml starting at 0 byte offset and it has a length of 252 bytes.

The other file generated by the export utility in this example is prodexport.del.001.xml which contains the XML content. Every XML data exported are concatenated and written to this file. Here is the content of the prodexport.del.001.xml file to give you a better idea.

File: prodexport.del.001.xml

```
<?xml version="1.0" encoding="UTF-8" ?><product pid="100-100-01"><description><n
ame>Snow Shovel, Basic 22"</name><details>Basic Snow Shovel, 22" wide, straight
handle with D-Grip</details><price>9.99</price><weight>1 kg</weight></descriptio
n></product><?xml version="1.0" encoding="UTF-8" ?><product pid="100-101-01"><de
scription><name>Snow Shovel, Deluxe 24"</name><details>A Deluxe Snow Shovel, 24
inches wide, ergonomic curved handle with D-Grip</details><price>19.99</price><w
eight>2 kg</weight></description></product><?xml version="1.0" encoding="UTF-8"
?><product pid="100-103-01"><description><name>Snow Shovel, Super Deluxe 26" Wid
e</name><details>Super Deluxe Snow Shovel, 26" wide, ergonomic battery heated cu
rved handle with upgraded D-Grip</details><price>49.99</price><weight>3 kg</weig
ht></description></product><?xml version="1.0" encoding="UTF-8" ?><product pid="
100-201-01"><description><name>Ice Scraper, Windshield 4" Wide</name><details>Ba
sic Ice Scraper 4" wide, foam handle</details><price>3.99</price></description><
/product>
```

Exporting XML data with XML options and modifiers

Like exporting large objects, you can specify the path(s) where the exported XML documents will go and the base filename of the output files. Consider the following example:

```
EXPORT TO prodexport.del DEL
    XML TO d:\xmlpath
    XMLFILE proddesc
    MODIFIED BY XMLINSEPFILES XMLNODECLARATION XMLCHAR
    XMLSAVESCHEMA
    MESSAGES msg.out
    SELECT * FROM product
```

Here, the relational data of the PRODUCT table is exported to the prodexport.del file. All XML data is then written in the directory specified in the XML TO clause, d:\xmlpath. The files with XML data are named proddesc.ext.xml where ext is a sequence number (e.g., proddesc.001.xml, proddesc.002.xml, proddesc.003.xml, etc.). This base filename is defined with the XMLFILE option.

You might also notice that a few modifiers are used in the example. Here is a summary of all the XML-related modifiers.

- XMLINSEPFILES causes the export utility to write each exported XML document to a separate XML file.

- XMLNODECLARATION indicates that the XML data is exported without an XML declaration tag. An XML declaration tag is by default written at the beginning of an XML document that includes an encoding attribute.

- XMLCHAR indicates that the XML data is written in the character codepage. By default, XML data is written out in Unicode. When this modifier is used, the value of the codepage file type modifier or the application codepage will be used instead.

- XMLGRAPHIC indicates that the exported XML data will be encoded in the UTF-16 codepage regardless of the codepage file type modifier or the application codepage. Note that XMLGRAPHIC is not used in this example.

The last option we are introducing here is XMLSAVESCHEMA. When an XML document was inserted, it can be validated against an XML schema. The XMLSAVESCHEMA option causes the export utility to also save the XML schema information for every exported XML data. A fully qualified SQL identifier of that schema will be stored as an SCH attribute inside the corresponding XML data specifier (XDS). Note that if the exported XML document was not validated against an XML schema or the schema object no longer exists in the database, an SCH attribute will not be included in the corresponding XDS.

The following shows you the result of the above export example.

File: prodexport.del

```
"100-100-01",,,,,,,"<XDS FIL='proddesc.001.xml' />"
"100-101-01",,,,,,,"<XDS FIL='proddesc.002.xml' SCH='DB2INST1.PRODUCT'/>"
"100-103-01",,,,,,,"<XDS FIL='proddesc.003.xml' SCH='DB2INST1.PRODUCT'/>"
"100-201-01",,,,,,,"<XDS FIL='proddesc.004.xml' />"
```

File: proddesc.001.xml

```
<product pid="100-100-01"><description><name>Snow Shovel, Basic 22"</name><detai
ls>Basic Snow Shovel, 22" wide, straight handle with D-Grip</details><price>9.99
</price><weight>1 kg</weight></description></product>
```

File: proddesc.002.xml

```
<product pid="100-101-01"><description><name>Snow Shovel, Deluxe 24"</name><deta
ils>A Deluxe Snow Shovel, 24 inches wide, ergonomic curved handle with D-Grip</d
etails><price>19.99</price><weight>2 kg</weight></description></product>
```

File: proddesc.003.xml

```
<product pid="100-103-01"><description><name>Snow Shovel, Super Deluxe 26" Wide<
/name><details>Super Deluxe Snow Shovel, 26" wide, ergonomic battery heated curv
ed handle with upgraded D-Grip</details><price>49.99</price><weight>3 kg</weight
></description></product>
```

File: proddesc.004.xml

```
<product pid="100-201-01"><description><name>Ice Scraper, Windshield 4" Wide</na
me><details>Basic Ice Scraper 4" wide, foam handle</details><price>3.99</price><
/description></product>
```

Exporting XML data with an XQuery

The `EXPORT` command also allows you to specify an XQuery statement so that the
export utility writes the result of an XQuery to an XML file. Let's examine the
following example.

```
EXPORT TO custexport.del DEL
    XML TO d:\xmlpath
    XMLFILE custphone
    MODIFIED BY XMLINSEPFILES XMLNODECLARATION
    MESSAGES msg.out
    SELECT XMLQUERY ('$doc/customerinfo/phone' PASSING INFO AS "doc") FROM customer
```

The XQuery in the above example returns all the phone numbers for every customer

stored in the CUSTOMER table under the XML column INFO. All the XML options and modifiers discussed apply to XQuery statements. Hence, this example will generate separate XML documents for each result of the XQuery. The files are located in d:\xmlpath and they are named custphone.ext.xml where ext is a sequence number. In addition, no XML declaration tag will be included in the documents.

Here is the content of one of the exported XML documents.

File: custphone.004.xml

```
<?xml version="1.0" encoding="UTF-8" ?><phone type="work">905-555-7258</phone><p
hone type="home">416-555-2937</phone><phone type="cell">905-555-8743</phone><pho
ne type="cottage">613-555-3278</phone>
```

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 such as the large objects and XML data. 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.
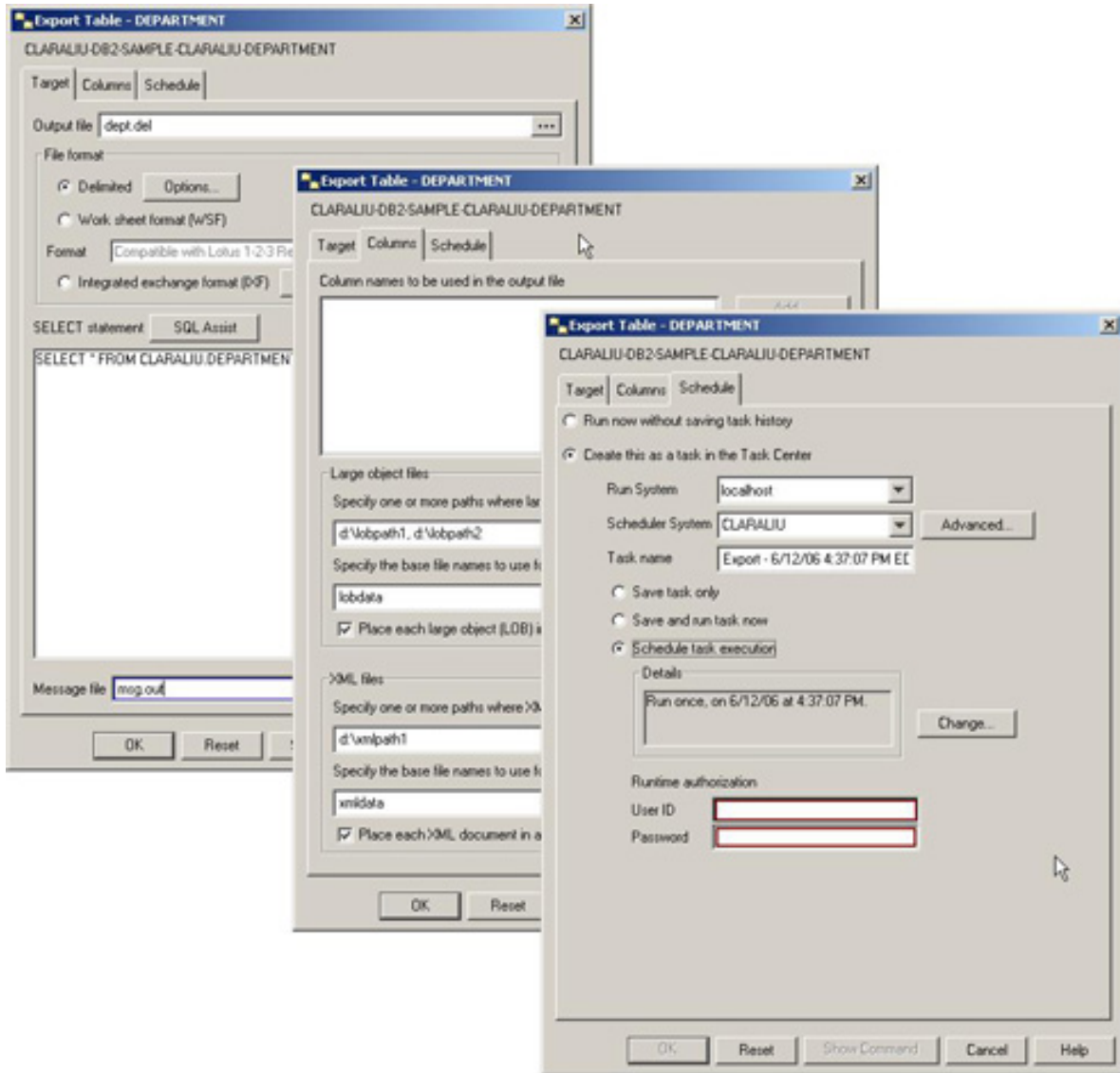
Section 4. DB2 IMPORT utility

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. To import data into a table that has protected rows and columns, you must have LBAC credentials that allow write access to all protected data in the table. In addition, importing to table with protected rows requires that your LBAC credentials is part of the security policy protecting the table.

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. You can also specify the `AUTOMATIC` option which allows the import internally determines when a commit needs to be performed. The utility will consider issuing a commit to avoid running into log full situation or to avoid lock escalation. 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 could use the message file to determine the last row that was successfully imported and committed. Then, you could restart the import with the `RESTARTCOUNT` option. Note that the behavior of the `SKIPCOUNT` option is the same as `RESTARTCOUNT`. 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 ROWCOUNT 100000
    MESSAGES msg.out
    INSERT INTO newtable
```

In the example, notice that the `ROWCOUNT` option is also used. It specifies the number of physical records to be imported. Because the `RESTARTCOUNT` option is used, the import utility will skip the first 30,000 records and import the next 100,000 records into the table.

By default, the import utility will acquire an exclusive lock on the target table before any rows are inserted. The exclusive lock is released as soon as the import completes. This is the behavior of the `ALLOW NO ACCESS` option. In order to allow concurrent applications to access the table data, you can use the `ALLOW WRITE ACCESS` option. Note that this option is not compatible with the `REPLACE`, `CREATE`, or `REPLACE_CREATE` import options. Here is an example of the `ALLOW WRITE ACCESS` option.

```
IMPORT FROM myfile.ixf OF IXF
    ALLOW WRITE ACCESS
    MESSAGES msg.out
    INSERT INTO newtable
```

Importing XML data

To import XML files, use the `XML FROM` option to specify one or more paths where XML files are stored. Otherwise, the import utility will look for the XML files in the current directory. You can choose how the XML documents are parsed; strip whitespace or preserve whitespace. If the `XMLPARSE` option is not specified, the parsing behavior for XML documents will be determined by the `CURRENT XMLPARSE OPTION` special register. Here is an example of the `XML FROM` and `XMLPARSE` options

```
IMPORT FROM myfile.ixf OF IXF
    XML FROM d:\xmlpath
    XMLPARSE PRESERVE WHITESPACE
    MESSAGES msg.out
    INSERT INTO newtable
```

When you insert or update an XML document, you might want to determine whether the structure, content, and data types of the XML document are valid. The import utility also supports XML validation through the use of the `XMLVALIDATE` option. There are three possible methods.

- `USING XDS` - Recall that you can export XML schema information and store it in the SCH attribute of the XML Data Specifier (XDS). The value of the SCH attribute will be used to perform validation. If there is no SCH attribute in the XDS, either the `DEFAULT`, `IGNORE`, or `MAP` will be considered.

- `USING SCHEMA` *schema-sqlid* - Use the XML schema specified in this clause.

- `USING SCHEMALOCATION HINTS` - Validate the XML documents against the schemas identified by the XML schema location hints in the source XML documents.

```
IMPORT FROM myfile.ixf OF IXF
    XML FROM d:\xmlpath
    XMLPARSE PRESERVE WHITESPACE
    XMLVALIDATE USING XDS
        DEFAULT S1.SCHEMA_A
        IGNORE (S1.SCHEMA_X, S1.SCHEMA_Y, S1.SCHEMA_Z)
        MAP (S1.SCHEMA_A, S1.SCHEMA_B)
    COMMITCOUNT 500 RESTARTCOUNT 30000
    MESSAGES msg.out
    INSERT INTO newtable
```

The above `IMPORT` command will:

- Insert data from myfile.ixf and XML files located in d:\xmlpath.

- Whitespace is preserved when the XML document is parsed.

- Each XML document is validated using the schema information identified in the SCH attribute of the XDS. However, if XDS for any particular row doesn't contain a SCH attribute, S1.SCHEMA_A will be used instead.

- For SCH attribute specified as S1.SCHEMA_X, or S1.SCHEMA_Y, or S1.SCHEMA_Z, validation will not be performed for the imported XML document.

- If the SCH attribute is specified as S1.SCHEMA_A, it will then mapped to S1.SCHEMA_B. Note that although the DEFAULT clause specifies S1.SCHEMA_A, any subsequent mapping will not be performed.

- The import utility will issue a commit after every 500 rows are imported.

- The import operation is started at record 30,001. The first 30,000 records are skipped.

- Any errors, warnings, and informational messages are written to the msg.out file.

- New data are inserted (or appended) into the *newtable*.

This example only gives you some idea of how the imported XML documents can be validated. There are more examples in the DB2 Information Center that demonstrate the power of the XMLVALIDATE option.

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.
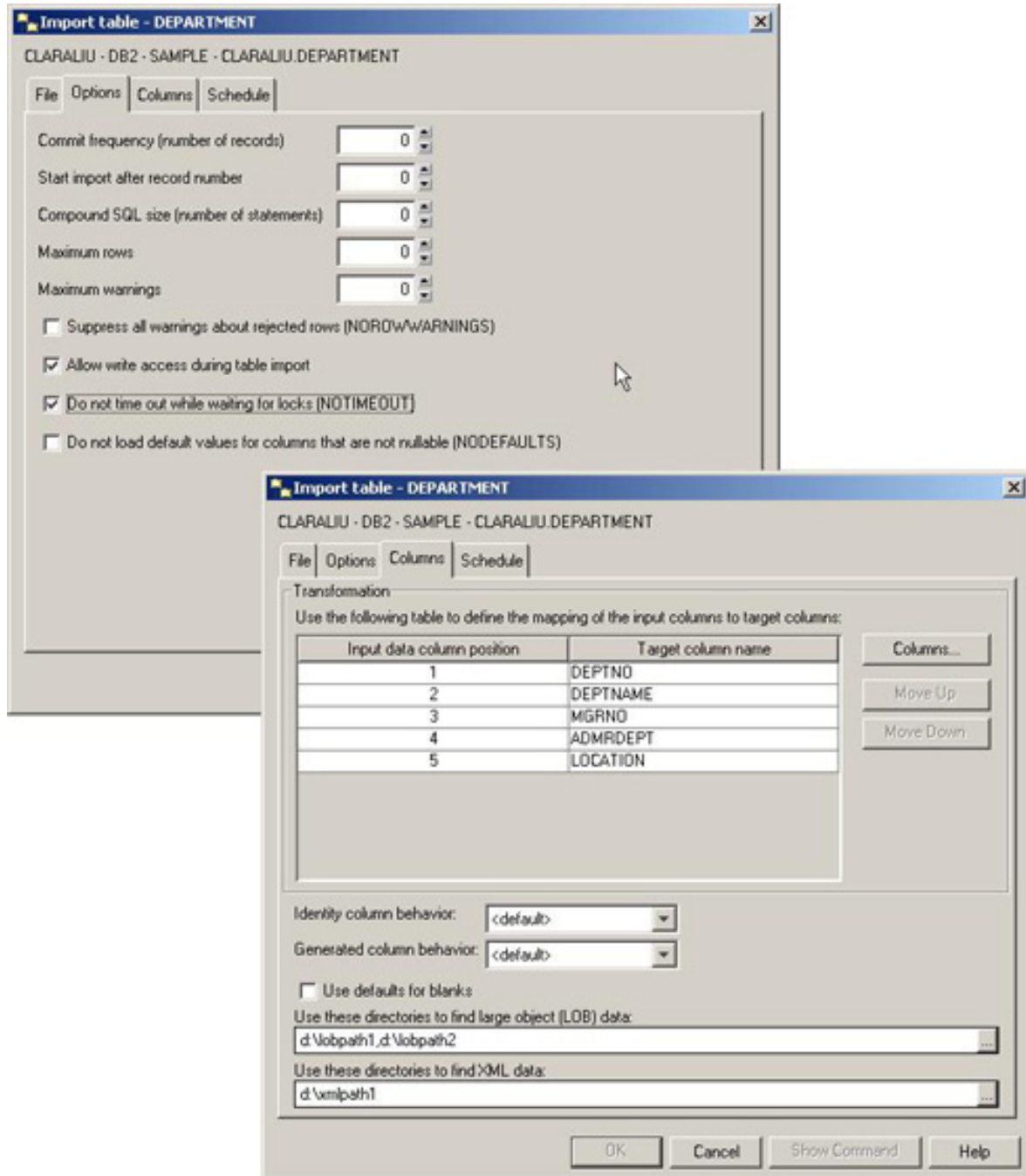
Section 5. DB2 LOAD utility

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. To load data into a table that has protected columns, you must have LBAC credentials that allow write access to all protected columns in the table. To load data into a table that has protected rows, you must have been granted a security label for write access that is part of the security policy protecting the table.

```
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 tab1;
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 tables with XML columns, 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. If STATISTICS USE PROFILE is specified, statistics is also collected according to the profile defined for the target table. This profile must be created before load is executed, otherwise a warning is returned and no statistics are collected.

3. **Delete phase:**

   - Deletes rows that caused unique key violations and places them in the exception table. Besides data 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

DB2 utilities
Page 19 of 48

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.
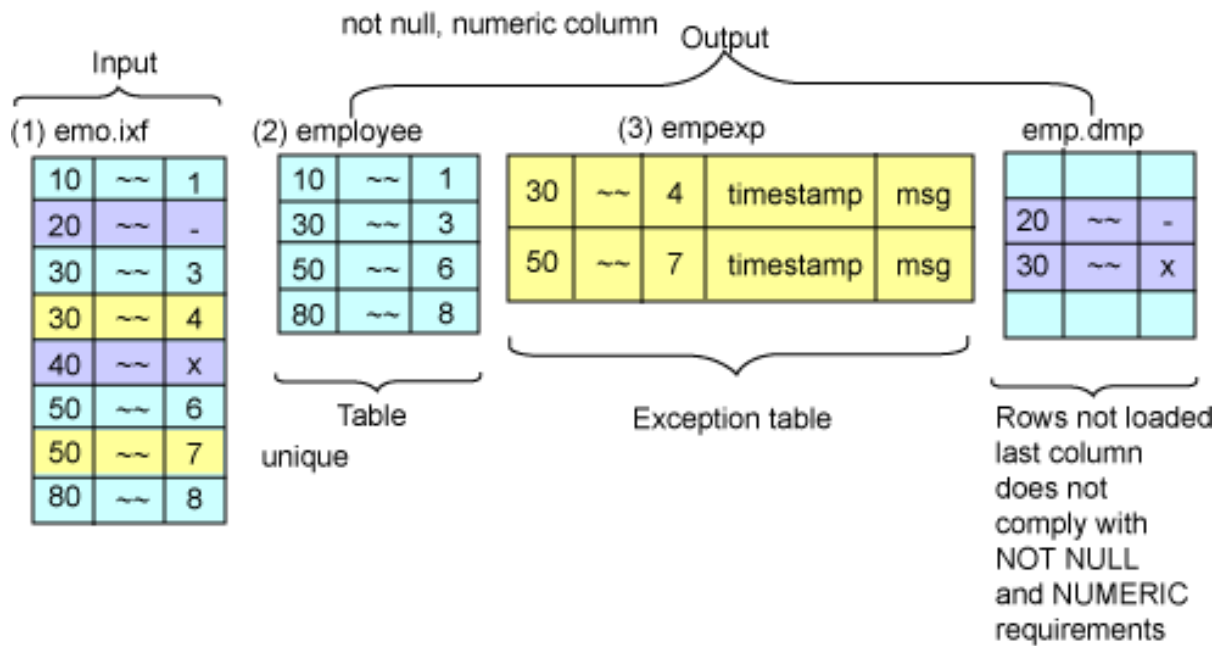
An *exception table* is a user-defined table that has to have the same column definition of the target table being loaded. If at least one of the columns is not present in the exception table, the offending rows will be discarded. Only two additional columns can be added to the end of the table: a timestamp column to record when a row is inserted, and a CLOB column to store the reason (or message) why the row is considered bad.

You'll have noticed that some of the concepts presented on this panel haven't been covered in detail yet. We will use some examples and put the pieces together in the rest of the section.

A load example

Let's look at an example to illustrate the steps involved in a load process:

```
LOAD FROM emp.ixf OF IXF
    MESSAGES msg.out
    MODIFIED BY DUMPFILE=c:\emp.dmp
    TEMPFILES PATH d:\tmp
    INSERT INTO employee
    FOR EXCEPTION empexp
```

not null, numeric column Output

Input

(1) emo.ixf   (2) employee   (3) empexp   emp.dmp

10	~~	1
20	~~	-
30	~~	3
30	~~	4
40	~~	x
50	~~	6
50	~~	7
80	~~	8

10	~~	1
30	~~	3
50	~~	6
80	~~	8

| 30 | ~~ | 4 | timestamp | msg |
| 50 | ~~ | 7 | timestamp | msg |

| 20 | ~~ | - |
| 30 | ~~ | x |

Table
unique

Exception table

Rows not loaded
last column
does not
comply with
NOT NULL
and NUMERIC
requirements

- In the diagram above, (1) shows the content of the input source file.

- The target table EMPLOYEE shown in (2) is created with the following column definition:

  - The first column must be unique.

  - The last column is a numeric column that cannot be NULL.

- An exception table, EMPEXP, shown in (3), is created with the same columns as EMPLOYEE as well as the timestamp and message columns.

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 increment ally. 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 he 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 USE PROFILE`: 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 in the build phase according to the profile defined for the target table.

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), 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.

Set Integrity 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 offended 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 `SET INTEGRITY 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 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          YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

- 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 SET INTEGRITY 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 SET INTEGRITY 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 `SET INTEGRITY 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 `LOGREATIN` 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 recreate 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.

LOAD using the Control Center

The Control Center provides easy-to-use graphical interfaces to perform load operations. As shown below, advanced options are described with hints. They can easily be set using the Control Center.
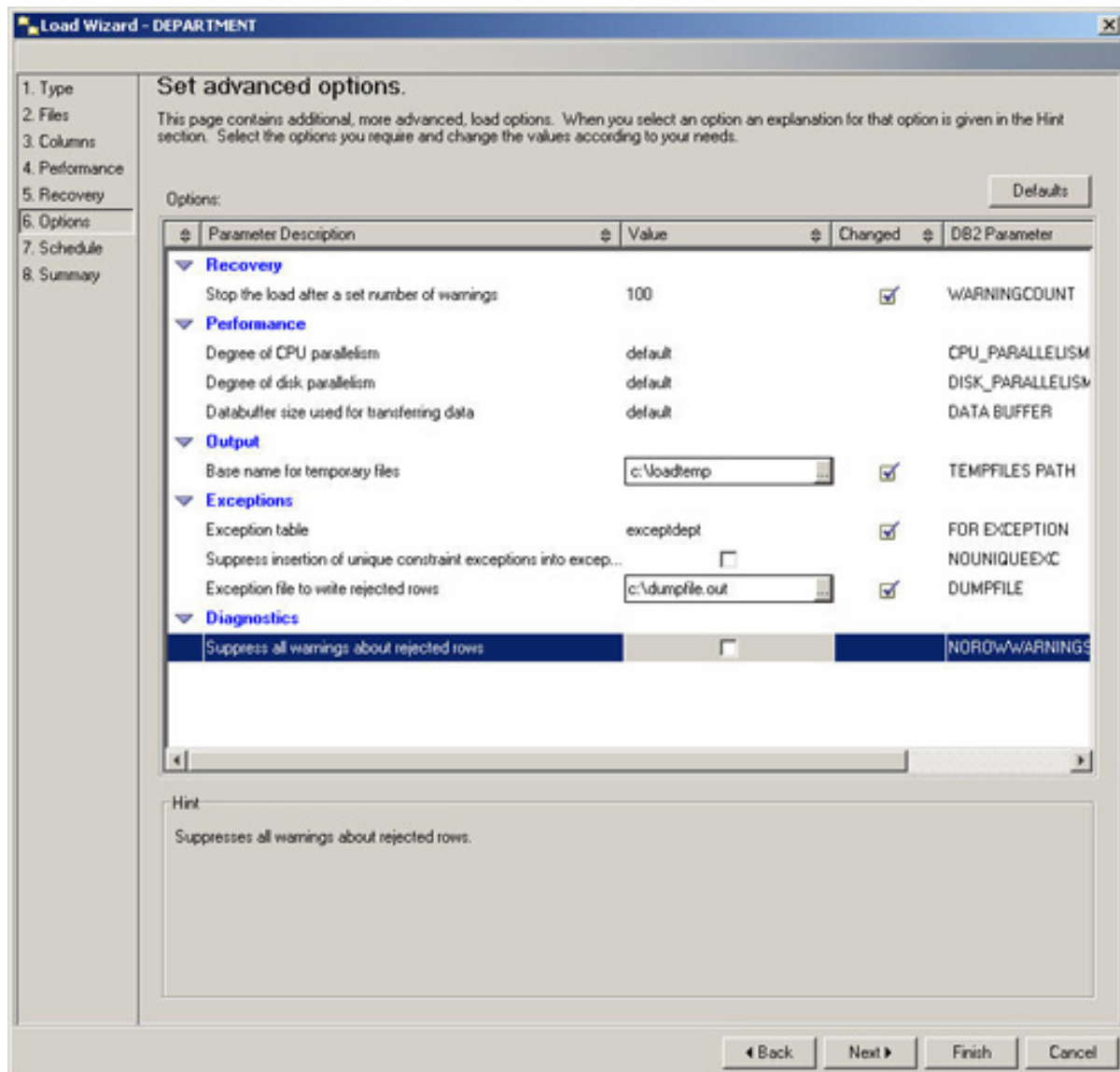
IMPORT vs. LOAD

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 no supported
Import into tabloes and views (Aliases supported)	Load tables only (Aliases supported)
No support for importing into materialized query tables	Supoort 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 keu is verified during load Other contraints are validated with the SET INTEGRITY command
If inturrupted, 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 RUNSTATA 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

Section 6. DB2 data movement utilities

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, LOAD, and COPY. The behavior of actions EXPORT, IMPORT, and LOAD is exactly the same as described in the previous panel. The only action you probably are not familiar with is COPY. It duplicates tables in a schema or schemas into a target database. The syntax of db2move is 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
```

Refer to the Command Reference to get a complete listing and descriptions of all the options. However, the `COPY` action warrant a discussion. With the `COPY` action, you specify one or more schemas with the `-sn` option. Only tables with exactly the same schema names specified in the `-sn` option will be copied (via export). If multiple schema names are specified, use commas to separate them and no blanks are allowed. Refer to the example below.

```
db2move sample COPY -sn db2inst1,prodschema -co TARGET_DB acctdb USER peter
USING petepasswd DDL_AND_LOAD
```

The above `db2move` command copies supported objects under the schema db2inst1 and prodschema. The `-co` option that follows makes the command more interesting. Option `TARGET_DB` specifies the target database which the schemas are going to be copied. This option is mandatory when `COPY` action is specified. In addition, the target database must be different from the source database. You may 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 will be created and tables will be populated in the target database. This is the behavior of the `DDL_AND_LOAD` mode. 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 may want to rename the schema when copying the objects to the target database. The `SCHEMA_MAP` option can be used to provide you for this purpose. You simply provide one or more pairs of schema mapping like this:

```
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, qualifed objects inside the object body remains unchanged. For example:

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

Schema rename from FOO to BAR will result in:

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

BAR.v1 created in the target database might fail if FOO.T1 is not defined.

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 above table space name mapping indicates that source TS1 is mapped to target TS2, 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
        NONRECOVERABLE
```

This command copies supported objects in the db2inst1 and prodschema from the SAMPLE database to the ACCTDB database. The authorization id peter and the associated password is used to connect to ACCTDB. The target tables already exist in ACCTDB and the tables will be 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 will be used instead.

NONRECOVERABLE option allows the user to use the table spaces that were loaded into immediately after the copy is completed. Backups of the table spaces are not required but highly recommended at the earlier convenient time.

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 illustrates. 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
```

The 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
```

The db2look is also capable of generating commands to register XML schemas. The following example generates the necessary REGISTER XMLSCHEMA and COMPLETE XMLSCHEMA commands (specified by option -xs) for objects with schema name db2inst1. The output db2look.sql will be created under /home/db2inst1 which is specified in the -xdir option.

```
db2look -d department -z db2inst1 -xs -xdir /home/db2inst1 -o db2look.sql
```

db2batch

Benchmarking is a process of evaluating the application in various aspects such as database response time, cpu and memory usage. Benchmark tests are based on a

repeatable environment so that the same test run under the same conditions. Results collected from the tests can then be evaulated and compared.

`db2batch` is a benchmarking tool that takes a set of SQL and/or XQuery statements, dynamically prepares, and describes the statements, and returns an answer set. Depending on the options used in the `db2batch` command, the answer set might return elapsed time of execution of the statements, database manager snapshots on memory usage such as bufferpool and cache information.

You can specify the statements you want to run benchmark on in a flat file or standard input. A number of control options can be set in the input file. They are specified with this syntax: `--#SET control_option value`. Here is an example of an input file with control options. For a complete listing of control options, please refer to the Information Center.

```
-- db2batch.sql
-- ------------
--#SET PERF_DETAIL 3
--#SET ROWS_OUT 5


-- This query lists employees, the name of their department
-- and the number of activities to which they are assigned for
-- employees who are assigned to more than one activity less than
-- full-time.
--#COMMENT Query 1
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2;
--#SET PERF_DETAIL 1
--#SET ROWS_OUT 5

--#COMMENT Query 2
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2;
```

- Option PERF_DETAIL 3 means that performance detail on elapsed time, a snapshot for the database manager, the database, and the application will be returned.

- Option ROWS_OUT 5 means that only 5 rows are to be fetched from the result set regardless of the actual number of rows returned for the query.

- COMMENT Query1 simply gives the statement a name, Query1.

The following command invokes the benchmark tool against the SAMPLE database with the input file `db2batch.sql`.

```
db2batch -d sample -f db2batch.sql
```

This command will return the result set of both queries limited to 5 rows, elapsed time, and CPU time of the queries. Database manager, database, and application snapshots are also returned. Since the output is quite large, we are only showing the summary of the `db2batch` command here.

```
* Summary Table:

Type       Number      Repetitions Total Time (s) Min Time (s)   ...
---------  ----------- ----------- -------------- --------------
Statement          1           1       0.052655       0.052655 ...
Statement          2           1       0.004518       0.004518 ...


...Max Time (s)   Arithmetic Mean Geometric Mean Row(s) Fetched Row(s) Output
   -------------  --------------- -------------- -------------- -------------
...      0.052655        0.052655       0.052655              5             5
...      0.004518        0.004518       0.004518              8             5


* Total Entries:             2
* Total Time:                0.057173 seconds
* Minimum Time:              0.004518 seconds
* Maximum Time:              0.052655 seconds
* Arithmetic Mean Time:      0.028587 seconds
* Geometric Mean Time:       0.015424 seconds
```

The `db2batch` command supports many options. We are just listing a few here for you to get an idea the power of the tool.

- -m *parameter_file* specifies an input file with parameter values to bind to the SQL statement parameter markers.

- -r *result_file* specifies an output file to contain the result of the command.

- -i *short|long|complete* specifies what is being measured for the elapsed time intervals. *short* measures the elapsed time to run each statement. *long* measures the elapsed time to run each statement including overhead between statements. *complete* measures the elapsed time to run each statement where the prepare, execute, and fetch times are reported separately.

- -iso specifies the isolation level used for the statement. By default, `db2batch` uses the Repeatable Read isolation level.

Section 7. 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 usage 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
```

You can be very specific when it comes to collecting statistics on the database objects. Different combination of the RUNSTATS options can be used to collect table statistics, index statistics, distribution statistics, sampling information, and etc. To simplify statistics collection, you can save the options you specify when you issue the RUNSTATS command in a statistics profile. If you want to collect the same statistics repeatedly on a table and do not want to re-type the command options, you simply do this:

```
RUNSTATS ON TABLE db2user.employee USE PROFILE
```

This command collects statistics on db2user.employee using the options recorded in the statistics profile for that table. So, how do you set a statistics profile? It is as easy as using the SET PROFILE ONLY option.

```
RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION ON COLUMNS ( empid, empname )
    SET PROFILE ONLY
```

Notice that the option will only set the profile, the RUNSTATS command will not run. If you need to modify a previously registered statistics profile, use the UPDATE PROFILE ONLY option. Similarly, this option will only update the profile without

running the `RUNSTATS` command. If you want to update the profile as well as updating the statistics, use the `UPDATE PROFILE` instead.

```
RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION DEFAULT NUM_FREQVALUES 50
 NUM_QUANTILES 50
    UPDATE PROFILE
```

`RUNSTATS` is a resource-intensive utility. However, in order to maintain efficient database operation, statistics must be collected regularly. You should find regular windows of reduced database activity so that database statistics can be collected without affect the database performance. In some environment, there is no such window. Throttling of `RUNSTATS` utility can be considered to limit the amount of resources consumed by the utility. When the database activity is low, the utility runs more aggressively. On the other hand, when the database activity increases, the resources allocated to executing `RUNSTATS` are reduced. Here is how to specify the level of throttling.

```
RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION DEFAULT NUM_FREQVALUES 50
 NUM_QUANTILES 50
    UTIL_IMPACT_PRIORITY 30
```

The acceptable priority value ranges from 1 to 100. 100 representing the highest priority (meaning unthrottled) and 1 representing the lowest. 50 is the default priority level.

Note that automatic statistics collection is enabled by default when the database is created. It can be turned off by setting the database configuration parameter AUTO_RUNSTATS to OFF.

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 both read and 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 fairly self-explanatory:

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

You can also reorganize an index. If the CLEANUP clause is used as shown in one of the examples below, a cleanup will be done instead of a reorganization.

```
REORG INDEX db2user.idxemp FOR TABLE db2user.employee ALLOW WRITE ACCESS
REORG INDEX db2user.idxemp FOR TABLE db2user.employee CLEANUP ONLY
```

REORGCHK is another data maintenance utility that has an option to retrieve current database statistics or 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: CLUSTERRATIO 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
SYSIBM      IBM83            -     -     -    -     -    -     -     -    -  -  -  -  -----
SYSIBM      IBM84            -     -     -    -     -    -     -     -    -  -  -  -  -----
SYSIBM      IBM85            -     -     -    -     -    -     -     -    -  -  -  -  -----
Table: SYSIBM.SYSBUFFERPOOLNODES
SYSIBM      IBM69            -     -     -    -     -    -     -     -    -  -  -  -  -----
Table: SYSIBM.SYSBUFFERPOOLS
SYSIBM      IBM67            1     1     0    1    22    0     1   100    -  -  0  0  -----
SYSIBM      IBM68            1     1     0    1    10    0     1   100    -  -  0  0  -----
Table: SYSIBM.SYSCHECKS
SYSIBM      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 recreate a package so that the current database statistics can be used. The command is very simple:

```
REBIND PACKAGE package_name
```

In many cases, SQL statements contain host variables, parameter markers, and special registers. The values of these variables are not know until run time. With the REOPT clause in the REBIND command, you can specify whether to have DB2

optimize an access path using real values for host variables, parameter markers, and special registers. There are three REOPT options:

- NONE - Real values of the host variables, parameter markers, and special registers used in the SQL statement will not be used to optimize an access path. The default estimates for these variable will be used instead.

- ONCE - The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers, or special registers when the query is first executed.

- ALWAYS - The access path for a give SQL statement will always be compiled and reoptimized using the values of the host variables, parameter markers, or special registers.

```
REBIND PACKAGE ACCTPKG REOPT ONCE
```

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

As for dynamic SQL statements, they are precompiled at runtime 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:

DB2 utilities
Page 37 of 48
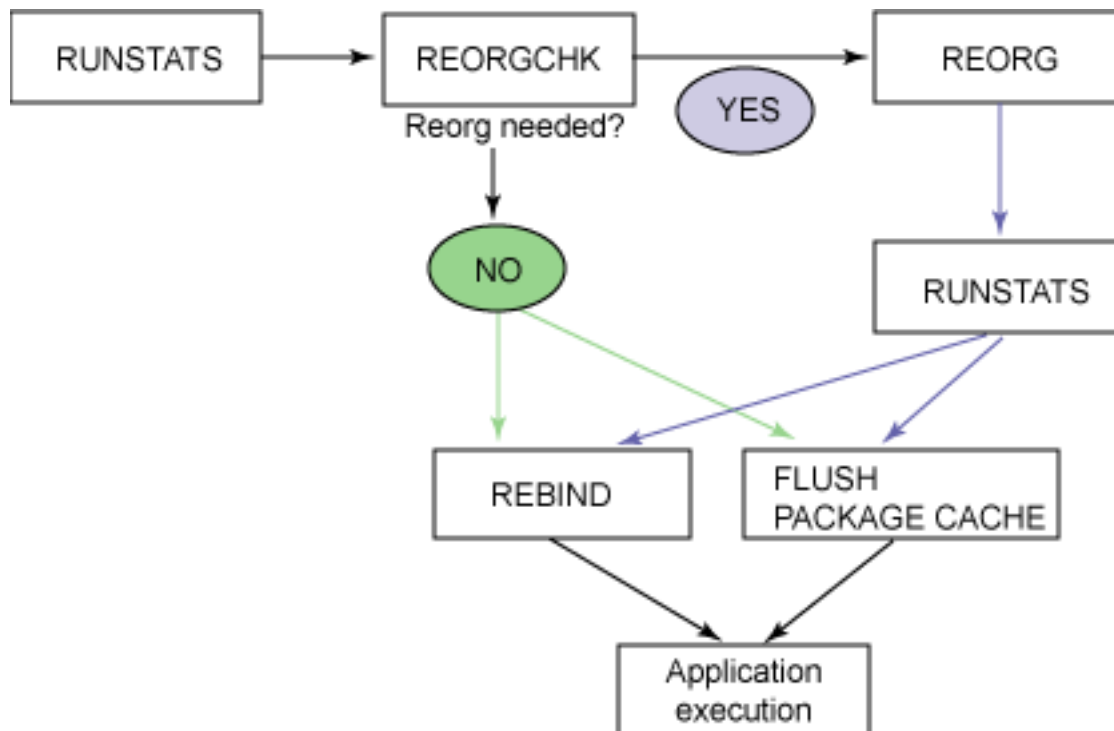
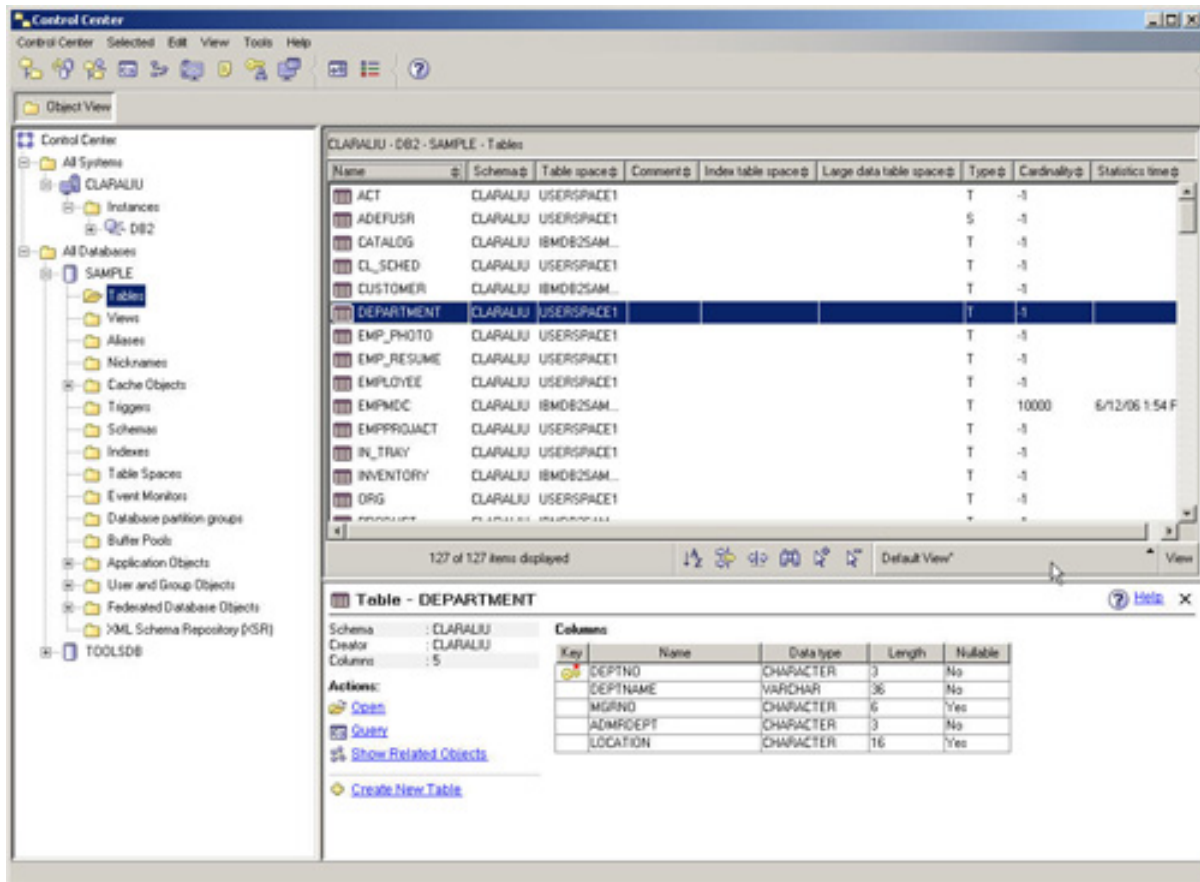## Section 8. DB2 Control Center

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). 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.
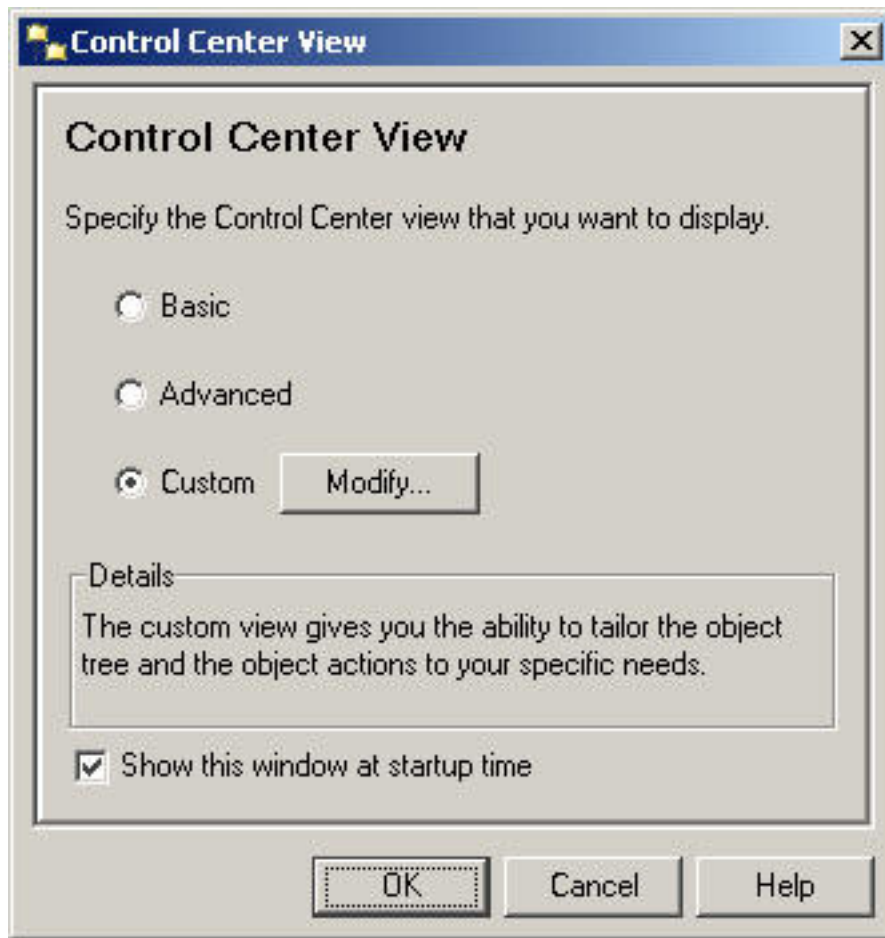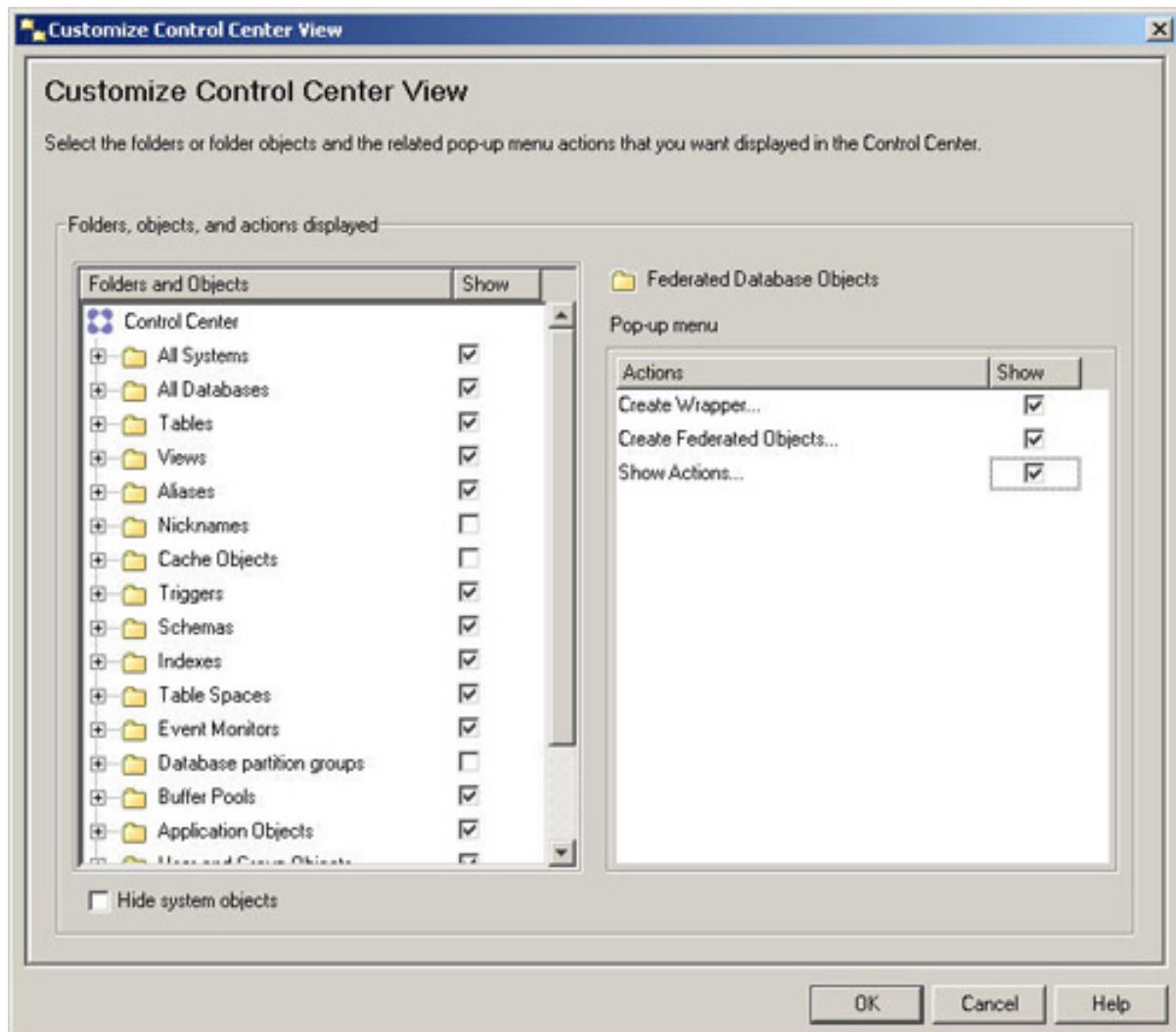

DB2 Control Center Views

DB2 9.1 allows you to select and customize Control Center views you want to work with. There are three different views:

- The basic view gives you an interface to work with all the essential objects such as databases, tables, and views.

- The advanced view displays all objects and actions available in the Control Center. This view is usually used if you want to connect to DB2 for z/OS or IMS systems.

- The custom view gives you the ability to customize the objects and actions according to your preference.

When you start the DB2 Control Center, you can specify which view you want to use just like what you see here.

If you choose **Modify** in the Control Center View, you will get to a window similar to the following. Notice that on the left pane, you choose the folders and objects to be displayed. According to the selection in this pane, the right pane allows you to further customize the associated actions (or pop-up menus).

Working with the DB2 Control Center

In the Control Center, information about any database object can easily be accessed. For example, if you select a table, details about that table will be retrieved and displayed in the right lower panel. To show related objects of this table, simply click the **Show Related Objects** link. As shown in the picture below, you can also launch other tools from the Tools menu.
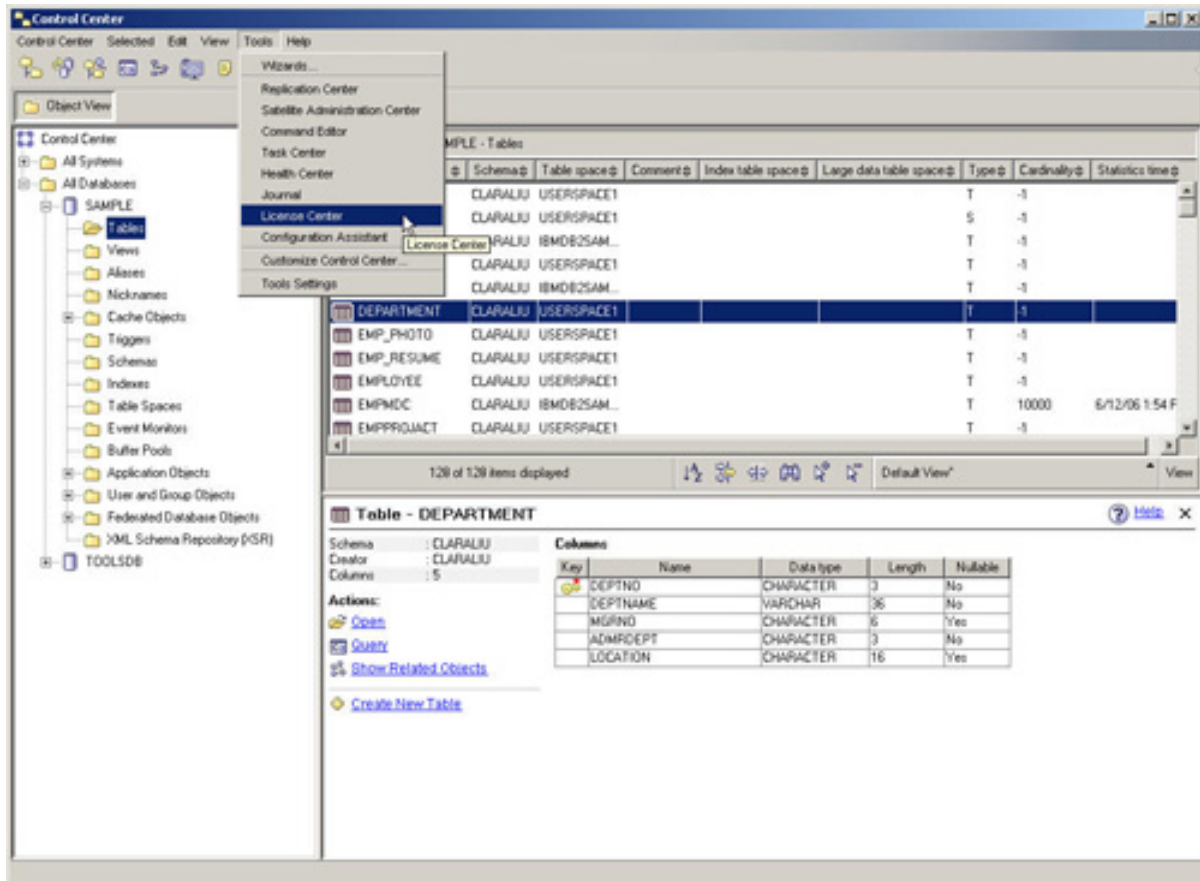
Section 9. DB2 Advisors

DB2 Design Advisor

The Design Advisor is a handy tool that can help you to determine what database objects can improve the performance of a given workload. A workload is basically a set of SQL statements for which the Design Advisor evaluates based on the characteristics of the workload, the characteristics of the database, and the hardware resources. It uses the DB2 optimizer, the database statistics, and the Explain mechanism to generate recommendations for new indexes, new materialized query tables (MQT), conversion to multidimensional clustering (MDC) tables, redistribution of tables, deletion of indexes and MQTs unused by the specified workload. You might be familiar with MQTs and MDCs. They are advanced database objects which are not covered in the DBA exam (731). If you are interested in advanced database objects, please refer to the DB2 Information Center.

You can start the advisor from the command line with the `db2advis` along with the necessary inputs. There are few ways to specify a workload using the `db2advis`

command.

You can specify a single SQL statement when issuing the db2advis command. The following example evaluates the given SQL statements and makes recommendations accordingly.

```
db2advis -d sample -s "select * from employee where workdept='A00' and
salary > 40000" -o output.out
```

You can use a set of dynamic SQL statements captured in a DB2 snapshot. To do so, you need to reset the database monitor with the command.

```
db2 reset monitor for database database-name
```

Let your application run for a desired period of time and allow DB2 snapshots to capture the dynamic SQL statements. Issue the following db2advis command to evaluate the workload and make recommendations. Here the -g option tells the design advisor to get the SQL statements from the dynamic SQL snapshots. In addition, the -p option causes the captured SQL statements to be stored in the ADVISE_WORKLOAD system tables.

```
db2advis -d sample -g -p -o output.out
```

Alternatively, you can create a workload file containing a set of SQL statements. Set the frequency of the statements in the workload. Here is a sample workload file.

```
--#SET FREQUENCY 100
SELECT COUNT(*) FROM EMPLOYEE;
SELECT * FROM EMPLOYEE WHERE LASTNAME='HAAS';
--#SET FREQUENCY 1
SELECT * FROM EMPLOYEE WHERE WORKDEPT='A00' AND SALARY > 40000;
```

Then, simply run the db2advis command with the -i option.

```
db2advis -d sample -i input.sql -o output.out
```

The Design Advisor can also be started from the Control Center. Select the database you want to work with, Select Design Advisor from its pop-up menu. You will get the Design Advisor as shown here.

DB2 utilities
Page 43 of 48
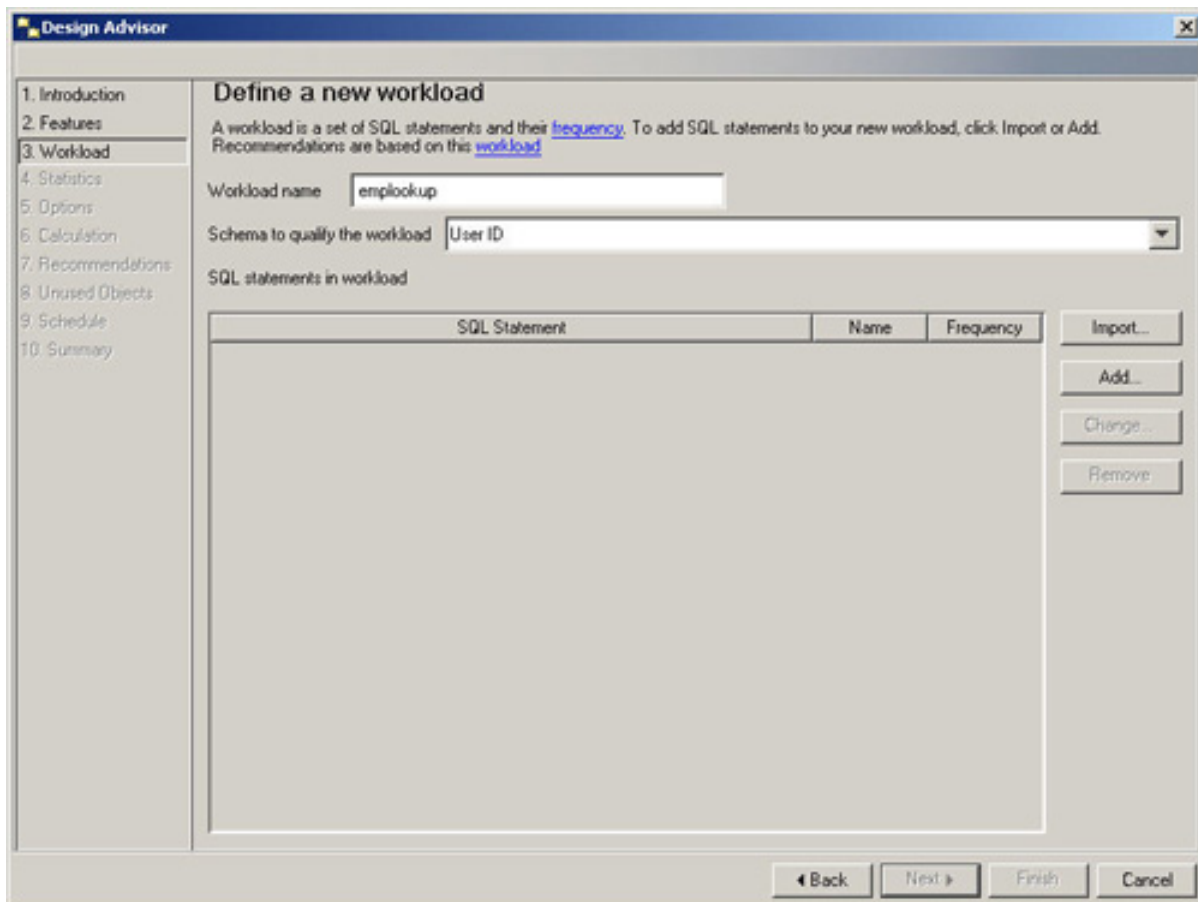
As you step through the tool, you can import a SQL workload, make changes to the SQL statements, update statistics for certain tables, choose to evaluate the workload now, and etc.
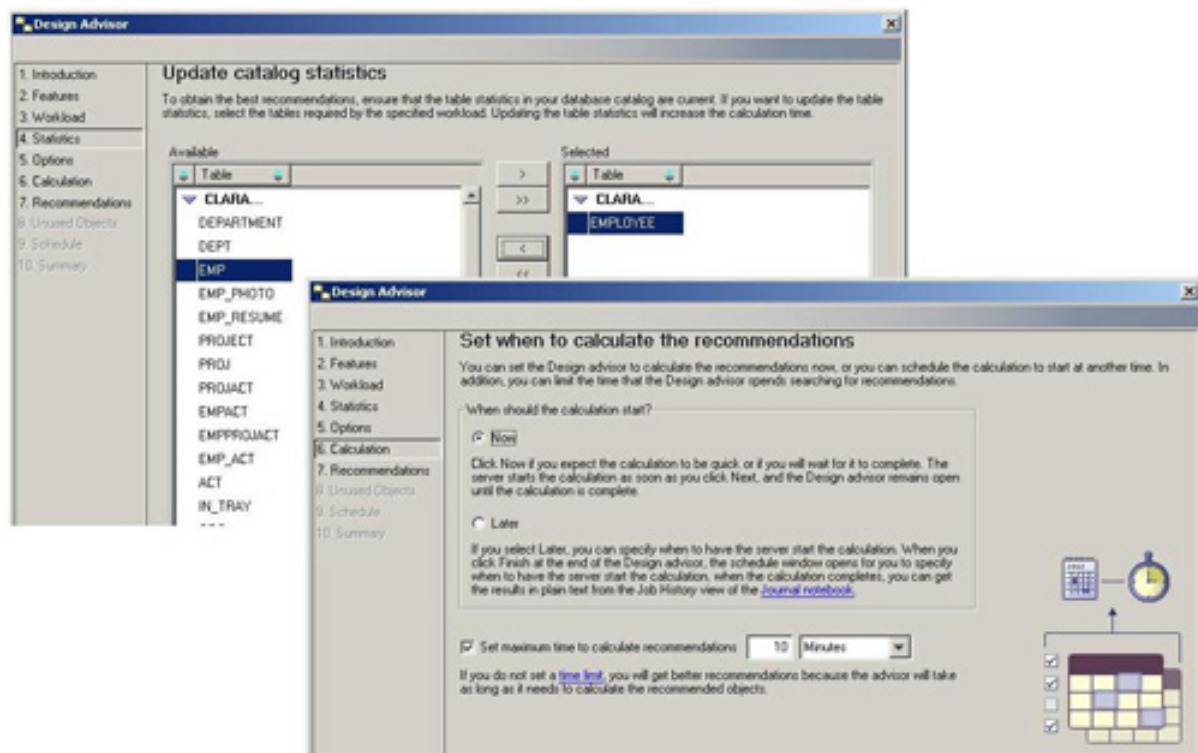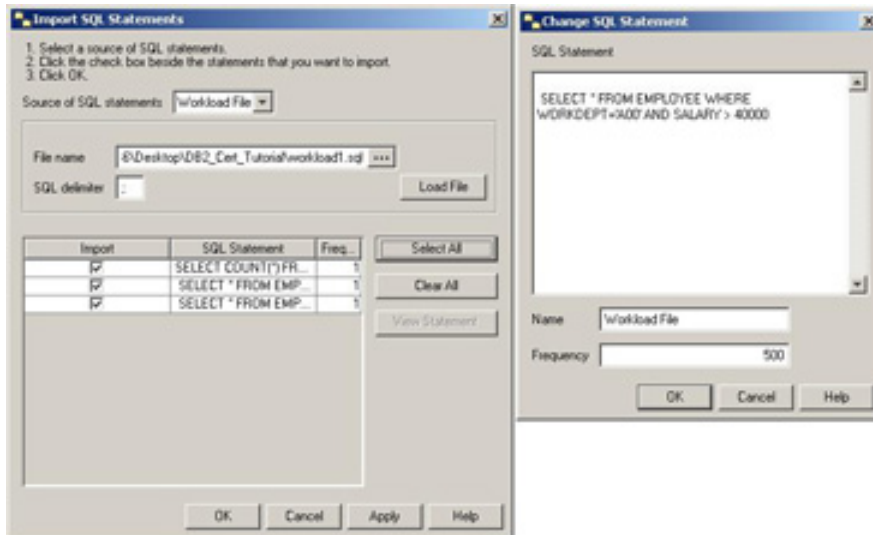
Section 10. Conclusion

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. Remember, XML data is also supported.

- The `IMPORT` utility, which can be used to perform bulk insert into a table or a view. Remember, XML data is also supported.

- 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 `SET INTEGRITY PENDING` state if the table has constraints other than unique constraints defined.

- The use of the `SET INTEGRITY` command to remove `SET INTEGRITY 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 few utilities such as `db2move`, `db2look`, `db2batch`, the Control Center and the Design Advisor.

Good luck with the exam!

Resources

Learn

- 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*

- For more on DB2 backup and restore strategies, please refer to the sixth tutorial in this series.

- Stay current with developerWorks technical events and webcasts.

- Check out the other parts of the DB2 9 DBA exam 731 prep tutorial series.

- You can learn more about managing DB2 from the following resources:

    - *DB2 9 Administration Guide: Performance*

    - DB2 9 Information Center, the following topics:

        - Setting up the DB2 server after manual installation

        - Configuring client to server communications

        - Configuring DB2 server communications

    - *DB2 9 Administration Guide: Implementation*:

- For more information on the DB2 9 for Linux, UNIX, and Windows Database Administration Certification (Exam 731), check out these links:

    - Information Management Training information

    - General Certification information -- including some book suggestions, exam objectives, and courses.

- DB2 9 overview. Find information about the new data server that includes patented pureXML technology.

- Certification exam site. Click the exam number to see more information about Exams 730 and 731.

Get products and technologies

- trial version of DB2 9 is available for free download.

- Download DB2 Express-C, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and

provides a solid base to build and deploy applications.

Discuss

- Participate in the discussion forum for this content.

About the author

Clara Liu
Clara Liu works as a DB2 UDB Product Planner at the IBM Toronto Laboratory. As a member of the DB2 product management team, she works closely with the DB2 marketing and development teams as well as customers and IBM business partners. She specializes in the area of database application development. Clara is also an active author in the DB2 for Linux, UNIX, and Windows community. She co-authored few IBM presses and one of her best sellers is *Understanding DB2 - Learning Visually with Examples* (Prentice Hall, 2004). You can reach Clara at claraliu@ca.ibm.com.