This study guide is designed to assist in learning about and finding additional information on the DB2 UDB Administration Fastpath Course for Version 8 (CT28D). Suggestions, additions, and comments are most welcome and should be directed to Dan Simchuk (simchuk@us.ibm.com) of the Information Management Partner Enablement Group.
# Table of Contents

DB2 Study Guide – FastPath Tutorial (DB2 v8) ................................................................. 0

Introduction ............................................................................................................................... 5

Overview of DB2 Universal Database.................................................................................. 5
  DB2 Family of Servers.......................................................................................................... 5
  Clients and Communication ............................................................................................... 6
  DB2 Major Elements........................................................................................................... 6
  Configuration Parameters................................................................................................. 7
  DB2 Hierarchy ..................................................................................................................... 8
  An Overview of Agents and Data I/O ............................................................................. 9
  Processor Configurations ................................................................................................. 10
  Memory Usage .................................................................................................................. 10

Getting Started with DB2 UDB GUIs ............................................................................. 12
  DB2UDB GUI Tools .......................................................................................................... 12

Creating Databases and Data Placement ...................................................................... 12

Creating Objects .................................................................................................................. 13
  Object Hierarchy ................................................................................................................ 13
  Data Types .......................................................................................................................... 14

Data Movement Utilities .................................................................................................. 15
  Export ................................................................................................................................. 15
    EXPORT Overview ........................................................................................................ 15
    The EXPORT Command ............................................................................................... 16
    EXPORT Exercise .......................................................................................................... 16
  Import ................................................................................................................................. 17
    IMPORT Overview ........................................................................................................ 17
    The IMPORT Command ............................................................................................... 18
    IMPORT Exercise .......................................................................................................... 18
  Load ................................................................................................................................... 19
    LOAD Overview ............................................................................................................ 19
    The Four Phases of the Load Process ........................................................................... 19
    The LOAD Command .................................................................................................. 20
    The LOAD Command (continued) ............................................................................... 21
    LOAD Exercise ............................................................................................................. 21

Loading Data in a Data Partitioned Environment ............................................................ 21
  Overview ........................................................................................................................... 21
  Concepts and Terminology ............................................................................................. 22
  Example of a DPF LOAD ................................................................................................. 22

Moving Data Between Systems ....................................................................................... 23
  Topics in Data Movement Utilities and Reference Guide Manual, Chapter 6 ............. 23

Recovery .............................................................................................................................. 24
  Recovery Concepts ......................................................................................................... 24
    Methods of Recovery .................................................................................................. 24
  Logging Introduction .................................................................................................... 24
  Logging Concepts .......................................................................................................... 25
Reorganizing with REORG .......................................................... 46
Statistical Catalog Views .......................................................... 46
Security ....................................................................................... 46
DB2 Security Overview ............................................................ 47
Authority Levels ...................................................................... 47
   Access Control Authority .................................................... 47
   Database Authority Summary .............................................. 48
Privileges ................................................................................... 49
   Authorities and Privileges .................................................. 49
   Controlling Use of Schemas ................................................. 50
   Protecting Resources through Programs .............................. 50
Explicit Privileges ..................................................................... 51
   GRANT TABLE/VIEW Privileges Support ............................ 51
   Public Privilege Support for Static SQL and Views ............... 52
Implicit Privileges .................................................................... 53

Trademark
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, Celeron, Intel Centrino, Intel Xeon, Itanium, Pentium and Pentium III Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linux Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.
Introduction
This study guide offers IBM Business Partners a roadmap for obtaining additional material and information on the topics covered in the FastPath Tutorial.

**Overview of DB2 Universal Database**

**DB2 family of servers**

*Quick Beginnings for DB2 Servers* – Chapters 1-3
Clients and communication
Quick Beginnings for DB2 Servers – Chapter 11

DB2 major elements
Administration Guide: Implementation – Chapter 1
Configuration parameters

Administration Guide: Performance – Chapter 13
**DB2 hierarchy**

*Administration Guide: Planning* – Chapter 1
An overview of agents and data I/O

Administration Guide: Performance – Chapter 2
**Processor configurations**

*Administration Guide: Planning* – Chapter 2

**Memory usage**

*Administrative Guide: Performance* – Chapter 8
Getting started with DB2 UDB GUIs

**DB2 UDB GUI tools**

Command Reference Manual

Creating databases and data placement

Command Reference Manual – Chapter 3
Creating objects
Administration Guide: Planning – Chapter 1

Object hierarchy
Data types
Data movement utilities

Data Movement Utilities and Reference Guide Manual – Chapters 1, 2, 3, 4, 6

Export

EXPORT overview

Export overview

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

The following information is required when exporting data:

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

You can also specify:

• New column names when exporting to IXF or WSF files. If you do not want to specify new column names, the column names in the existing table or view are used in the exported file.
• Additional options to customize the export operation.
• A message file name. During DB2 operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter. These message files are standard ASCII text files. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

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

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

EXPORT TO—filename—OF—filetype—LOBS TO—lob-path

LOBFILE—filename MODIFIED BY—filetype-mod

METHOD N—(column-name—)

select-statement

HIERARCHY—STARTING—sub-table-name traversal-order-list

where-clause

traversal-order-list:

(—sub-table-name—)

EXPORT exercise

Assignment – Export the EMPLOYEE table from the SAMPLE database to a delimited file that you can then read into a spreadsheet. Try this first by using the Export Notebook in the Control Center, then using the CLP.

Hints:

- The SQL select-statement to use will look something like:
  
  SELECT * FROM EMPLOYEE

- Your character delimiter will be the default (") – the double quote mark

- Your column delimiter will be the default (,) – comma

- Your filename can be any single word you’d like to use

- Your filetype will be DEL

- The final command will look like:

  EXPORT TO Employee OF DEL SELECT * FROM EMPLOYEE
**Import**

**IMPORT overview**

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

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

The following information is required when importing data:

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

You can also specify:

- The method to use for importing the data: column location, column name, or relative column position.
- The number of rows to INSERT before committing the changes to the table. Requesting periodic COMMITs reduces the number of rows that are lost if a failure and a ROLLBACK occur during the import operation. It also prevents the DB2 logs from getting full when processing a large input file.
- The number of file records to skip before beginning the import operation. If an error occurs, you can restart the import operation immediately following the last row that was successfully imported and committed.
- The names of the columns within the table or view into which the data is to be inserted.
- A message file name. During DB2 operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter. These message files are standard ASCII text files. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.
The IMPORT command

The statement will look something like this:

```sql
DB2 IMPORT FROM employee.del OF DEL INSERT INTO employee2
```

To make the table (IMPORT does not know about the DDL needed), try

```sql
CREATE TABLE employee2 LIKE employee
```
**Load**

**LOAD overview**

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data. The utility can handle most data types, including large objects (LOBs) and user-defined types (UDTs). The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs. The load utility does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes). The load process consists of four distinct phases.

**The four phases of the load process**

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

- **Load**, during which the data is written to the table. During the load phase, data is loaded into the table, and index keys and table statistics are collected, if necessary. Save points, or points of consistency, are established at intervals specified through the SAVECOUNT parameter in the LOAD command. Messages are generated, indicating how many input rows were successfully loaded at the time of the save point. For DATALINK columns defined with FILE LINK CONTROL, link operations are performed for non-NULL column values. If a failure occurs, you can restart the load operation; the RESTART option automatically restarts the load operation from the last successful consistency point. The TERMINATE option rolls back the failed load operation.

- **Build**, during which indexes are produced. During the build phase, indexes are produced based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (if the STATISTICS YES with INDEXES option was specified). The statistics are similar to those collected through the RUNSTATS command. If a failure occurs during the build phase, the RESTART option automatically restarts the load operation at the appropriate point.

- **Delete**, during which the rows that caused a unique key violation or a DATALINK violation are removed from the table. Unique key violations are placed into the exception table, if one was specified, and messages about rejected rows are written to the message file. Following the completion of the load process, review these messages, resolve any problems, and insert corrected rows into the table. Do not attempt to delete or to modify any temporary files created by the load utility. Some temporary files are critical to the delete phase. If a failure occurs during the delete phase, the RESTART option automatically restarts the load operation at the appropriate point. **Note:** Each deletion event is logged. If you have a large number of records that violate the uniqueness condition, the log could fill up during the delete phase.

- **Index copy**, during which the index data is copied from a system temporary table space to the original table space. This will only occur if a system temporary table space was specified for index creation during a load operation with the READ ACCESS option specified.

The following information is required when loading data:

- The path and the name of the input file, named pipe, or device.
- The name or alias of the target table.
• The format of the input source. This format can be DEL, ASC, PC/IXF, or CURSOR.
• Whether the input data is to be appended to the table, or is to replace the existing data in the table.
• A message file name, if the utility is invoked through the application programming interface (API), db2Load.

The LOAD command
The LOAD command (continued)

LOAD exercise
Use the LOAD command to take the output file from the EXPORT exercise, and create a new table named EMPLOYEE3 in the SAMPLE database.

Loading data in a data partitioned environment

Overview
In a partitioned database, large amounts of data are located across many partitions. Partitioning keys are used to determine on which database partition each portion of the data resides. The data must be partitioned before it can be loaded at the correct database partition. When loading tables in a partitioned database environment, the load utility can:

- Partition input data in parallel.
- Load data simultaneously on corresponding database partitions.
- Transfer data from one system to another system.
- Partitioned database load operations take place in 2 phases: A setup phase, where partition resources such as table locks are acquired, and a load phase where the data is loaded into the partitions. You can use the ISOLATE_PART_ERRS option of the LOAD command to select how errors will be handled during either of these phases, and how errors on one or more of the partitions will affect the load operation on the partitions that are not experiencing errors.

When loading data into a partitioned database you can use one of the following modes:

- PARTITION_AND_LOAD. Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- PARTITION_ONLY. Data is partitioned (perhaps in parallel) and the output is written to files in a specified location on each loading partition. Each file includes a partition header that specifies how the data was partitioned, and that the file can be loaded into the database using the LOAD_ONLY mode.
- LOAD_ONLY. Data is assumed to be already partitioned; the partition process is skipped, and the data is loaded simultaneously on the corresponding database partitions.
- LOAD_ONLY_VERIFY_PART. Data is assumed to be already partitioned, but the data file does not contain a partition header. The partitioning process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct partition. Rows containing partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If partition violations exist on a particular loading partition, a single warning will be written to the load message file for that partition.
- ANALYZE. An optimal partitioning map with even distribution across all database partitions is generated.
Concepts and terminology

The following terminology will be used when discussing the behavior and operation of the load utility in a partitioned database environment:

- The coordinator partition is the database partition to which the user connects to perform the load operation. In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, it is assumed that the data file resides on this partition unless the CLIENT option of the load command is specified. Specifying the CLIENT option of the load command indicates that the data to be loaded resides on a remotely connected client.

- In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, the pre-partitioning agent reads the user data and distributes it in round-robin fashion to the partitioning agents which will partition the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per partition for any load operation.

- In the PARTITION_AND_LOAD, LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, load agents run on each output partition and coordinate the loading of data to that partition.

- Load to file agents run on each output partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their partition.

- A file transfer command agent runs on the coordinator partition and is responsible for executing a file transfer command.

Example of a DPF LOAD

LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1 PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
Moving data between systems

Topics in Data Movement Utilities and Reference Guide Manual, Chapter 6

- "Moving Data Across Platforms - File Format Considerations"
- "Moving Data With DB2 Connect" on page 285
- "db2move - Database Movement Tool" on page 287
- "db2relocatedb - Relocate Database" on page 292
- "Moving Data Between Typed Tables" on page 293
- "Using Replication to Move Data" on page 298
- "Using the Data Warehouse Center to Move Data" on page 300.
- "Moving Data Using the Cursor File Type" on page 302.
Recovery

Data Recovery and High Availability Guide and Reference – Chapters 1-4

Recovery concepts

Methods of recovery

Logging introduction
Logging concepts

Circular logging

Archival logging
**Logging configuration parameters**

```
| Primary log files (logprimary) | ? | ? | New log path (newlogpath) |
| Secondary log files (logrecord) | ? | + | Mirror log path (mirrlogpath) |
| Log Size (logfilesz) | ? | N | Log retain (logretain) |
| Log Buffer (logbufsz) | Y | E | User exit (userexit) |
| Number of Commits to Group (mincommit) | N | Q | Log Records to Write Before Soft Checkpoint (softmax) |
```

**Recovery history file**

![Diagram showing Recovery history file](image)

**Backup, restore and rollforward**

*Data Recovery and High Availability Guide and Reference*

All backup, restore, and roll-forward operations should only be performed with all connections to the database closed. Once any of the operations complete, a TERMINATE is issued that will eliminate all existing connections.

**BACKUP DATABASE command**

Creates a backup copy of a database or a table space.

If a connection to the specified database already exists, that connection will be terminated and a new connection established specifically for the backup operation. The connection is terminated at the completion of the backup operation.
Typical BACKUP script (Non-Partitioned)

```
db2 backup database sample to /home/db2inst1/backup
```

Typical BACKUP script (Partitioned)

```
db2_all "db2 backup database sample to /home/db2inst1/backup"
```

Make sure there is a "/home/db2inst1/backup" on each node.
**RESTORE DATABASE command**

![Diagram showing the RESTORE DATABASE command process]

The image illustrates the process of restoring a database from a remote location to a local one. The diagram shows the command being executed, with the database being restored from a remote source to a local database named MUSICDB.
Typical RESTORE script (Non-Partitioned)

```bash
# Restore SAMPLE Database
# Note that DB cfg parms are saved in backup image, so if parms change, backup
# should be restored, parm(s) changed, new backup taken, and timestamp changed
# in this script
###timestamp is not needed if only one backup image in directory
###IMAGETIME=20031009......
echo
echo ... "If the following message appears, it's OK (means that existing DB was
overwritten):"
echo ... SQ2540W Restore is successful, however a warning "2539" was encountered
echo ... during Database Restore while processing in No Interrupt mode.
echo
date
debug
```
Typical RESTORE script (Partitioned)

```bash
# Restore SAMPLE Database
# Note that DB cfg parms are saved in backup image, so if parms change, backup
# should be restored, parm(s) changed, new backup taken, and timestamp changed
# in this script
###timestamp is not needed if only one backup image in directory
###IMAGETIME=20031009......
echo
echo ... "If the following message appears, it's OK (means that existing DB was overwritten):"
echo ... SQ2540W  Restore is successful, however a warning "2539" was encountered
echo ... during Database Restore while processing in No Interrupt mode.
echo
date
# restore catalog node (0) separately, since it has to run first anyway
db2_all "<<<+0< db2 -v restore database sample from /home/db2inst1/backup replace existing without prompting"
date
###db2_all "db2 -v restore database sample from ...... taken at $IMAGETIME replace existing without prompting"
```

**ROLLFORWARD DATABASE command**

Recovers a database by applying transactions recorded in the database log files. Invoked after a database or a table space backup image has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, either logretain, userexit, or both of these database configuration parameters must be enabled) before the database can be rollforward recovered.

In a partitioned database environment, this command can only be invoked from the catalog partition. A database or table space rollforward operation to a specified point in time affects all partitions that are listed in the db2nodes.cfg file. A database or table space rollforward operation to the end of logs affects the partitions that are specified. If no partitions are specified, it affects all partitions that are listed in the db2nodes.cfg file; if rollforward recovery is not needed on a particular partition, that partition is ignored.
Typical ROLLFORWARD DATABASE command

```
db2 rollforward database sample to end of logs
```
**Table space backup/restore concepts**

- Roll forward must be enabled
- Restore subset of table spaces
- Multiple table spaces in image may be desirable
  - Ease of table recovery strategy
  - Access to related tables managed coherently
- Long field/LOB data and REORG
- Point in time recovery supported

**Monitoring and problem determination**

*System Monitor Guide and Reference*

- Snapshot Monitor
  - Query Operational Database Status for an INSTANT in Time
- Event Monitor
  - Query Operational Status OVER Time for a Specific Activity
- Health Monitor
  - Gauges the health of the database manager or database performance
  - Can be accessed from Health Center, Web Health Center, CLP, APIs
Concurrency
Administration Guide: Performance – Chapter 3

Concurrency issues
Because many users access and change data in a relational database, the database manager must be able both to allow users to make these changes and to ensure that data integrity is preserved. Concurrency refers to the sharing of resources by multiple interactive users or application programs at the same time. The database manager controls this access to prevent undesirable effects, such as:

Lost updates
Two applications, A and B, might both read the same row from the database and both calculate new values for one of its columns based on the data these applications read. If A updates the row with its new value and B then also updates the row, the update performed by A is lost.

Access to uncommitted data
Application A might update a value in the database, and application B might read that value before it was committed. Then, if the value of A is not later committed, but backed out, the calculations performed by B are based on uncommitted (and presumably invalid) data.

Non-repeatable reads
Some applications involve the following sequence of events: application A reads a row from the database, then goes on to process other SQL requests. In the meantime, application B either modifies or deletes the row and commits the change. Later, if application Attempts to read the original row again, it receives the modified row or discovers that the original row has been deleted.

Phantom read phenomenon
The phantom read phenomenon occurs when:
1. Your application executes a query that reads a set of rows based on some search criterion.
2. Another application inserts new data or updates existing data that would satisfy your application’s query.
3. Your application repeats the query from step 1 (within the same unit of work).
Some additional (“phantom”) rows are returned as part of the result set that were not returned when the query was initially executed (step 1).
**Locking**

<table>
<thead>
<tr>
<th>IN</th>
<th>Intent Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Intent Share</td>
</tr>
<tr>
<td>IX</td>
<td>Intent Exclusive</td>
</tr>
<tr>
<td>S</td>
<td>Share with Intent X</td>
</tr>
<tr>
<td>S</td>
<td>Share</td>
</tr>
<tr>
<td>U</td>
<td>Update</td>
</tr>
<tr>
<td>X</td>
<td>Exclusive</td>
</tr>
<tr>
<td>Z</td>
<td>Superexclusive</td>
</tr>
</tbody>
</table>

Row Locking also used

<table>
<thead>
<tr>
<th>Minimum Supporting Table Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>W</td>
</tr>
<tr>
<td>NS</td>
</tr>
<tr>
<td>NW</td>
</tr>
</tbody>
</table>

**An Application does not acquire**
- Row locks

**If it is using Table Locks of**
- S, U, X, or Z
### Table Locks

<table>
<thead>
<tr>
<th>Mode of Lock A</th>
<th>IN</th>
<th>IS</th>
<th>S</th>
<th>IX</th>
<th>SIX</th>
<th>U</th>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>IS</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>IX</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>SIX</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>U</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Z</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

### Row Locks

<table>
<thead>
<tr>
<th>Mode of Lock A</th>
<th>S</th>
<th>U</th>
<th>X</th>
<th>W</th>
<th>NS</th>
<th>NW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>U</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>X</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>W</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>NS</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>NW</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

### State of Held Resource

<table>
<thead>
<tr>
<th>State Being Requested</th>
<th>none</th>
<th>IN</th>
<th>IS</th>
<th>NS</th>
<th>S</th>
<th>IX</th>
<th>SIX</th>
<th>U</th>
<th>X</th>
<th>Z</th>
<th>NW</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>IN</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>IS</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>NS</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>S</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>IX</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>SIX</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>U</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>X</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Z</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>NW</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>W</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
**Isolation levels**

**Performance impact of isolation levels**

An *isolation level* determines how data is locked or isolated from other processes while the data is being accessed. The isolation level will be in effect for the duration of the unit of work. Applications that use a cursor declared with a `DECLARE CURSOR` statement using the `WITH HOLD` clause will keep the chosen isolation level for the duration of the unit of work in which the `OPEN CURSOR` was performed. DB2® supports the following isolation levels:

- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read.

Detailed explanations for each of the isolation levels follows in decreasing order of performance impact, but in increasing order of care required when accessing and updating data.

**Repeatable read**

*Repeatable Read* (RR) locks all the rows an application references within a unit of work. Using Repeatable Read, a `SELECT` statement issued by an application twice within the same unit of work in which the cursor was opened, gives the same result each time. With Repeatable Read, lost updates, access to uncommitted data, and phantom rows are not possible.

The Repeatable Read application can retrieve and operate on the rows as many times as needed until the unit of work completes. However, no other applications can update, delete, or insert a row that would affect the result table, until the unit of work completes. Repeatable Read applications cannot see uncommitted changes of other applications.

With Repeatable Read, every row that is referenced is locked, not just the rows that are retrieved. Appropriate locking is performed so that another application cannot insert or update a row that would be added to the list of rows referenced by your query, if the query was re-executed. This prevents phantom rows from occurring. For example, if you scan 10,000 rows and apply predicates to them, locks are held on all 10,000 rows, even though only 10 rows qualify.

**Note:** The Repeatable Read isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking are used. Since Repeatable Read may acquire and hold a considerable number of locks, these locks may exceed the number of locks available as a result of the `locklist` and `maxlocks` configuration parameters. In order to avoid lock escalation, the optimizer may elect to acquire a single table-level lock immediately for an index scan, if it believes that lock escalation is very likely to occur. This functions as though the database manager has issued a `LOCK TABLE` statement on your behalf. If you do not want a table-level lock to be obtained ensure that enough locks are available to the transaction or use the Read Stability isolation level.

**Read stability**

*Read Stability* (RS) locks only those rows that an application retrieves within a unit of work. It ensures that any qualifying row read during a unit of work is not changed by other application processes until the unit of work completes, and that any row changed by another application process is not read until the change is committed by that process. That is, “non-repeatable read” behavior is **not** possible.

Unlike repeatable read, with Read Stability, if your application issues the same query more than once, you may see additional *phantom rows* (the *phantom read phenomenon*). Recalling the example of scanning 10,000 rows, Read Stability only locks the rows that qualify. Thus, with Read Stability, only 10 rows are retrieved, and a lock is held only on those ten rows. Contrast this with Repeatable Read, where in this example, locks would be held on all 10,000 rows. The locks that are held can be share, next share, update, or exclusive locks.
Note: The Read Stability isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking are used. One of the objectives of the Read Stability isolation level is to provide both a high degree of concurrency as well as a stable view of the data. To assist in achieving this objective, the optimizer ensures that table level locks are not obtained until lock escalation occurs.

The Read Stability isolation level is best for applications that include all of the following:

- Operate in a concurrent environment
- Require qualifying rows to remain stable for the duration of the unit of work
- Do not issue the same query more than once within the unit of work, or do not require that the query get the same answer when issued more than once in the same unit of work.

Cursor stability

Cursor Stability (CS) locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.

No other applications can update or delete a row that a Cursor Stability application has retrieved while any updatable cursor is positioned on the row. Cursor Stability applications cannot see uncommitted changes of other applications.

Recalling the example of scanning 10,000 rows, if you use Cursor Stability, you will only have a lock on the row under your current cursor position. The lock is removed when you move off that row (unless you update that row).

With Cursor Stability, both non-repeatable read and the phantom read phenomenon are possible. Cursor Stability is the default isolation level and should be used when you want the maximum concurrency while seeing only committed rows from other applications.

Uncommitted read

Uncommitted Read (UR) allows an application to access uncommitted changes of other transactions. The application also does not lock other applications out of the row it is reading, unless the other application attempts to drop or alter the table. Uncommitted Read works differently for read-only and updatable cursors.

Read-only cursors can access most uncommitted changes of other transactions. However, tables, views, and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes by other transactions can be read before they are committed or rolled back.

Note: Cursors that are updatable operating under the Uncommitted Read isolation level will behave as if the isolation level was cursor stability. When it runs a program using isolation level UR, an application can use isolation level CS. This happens because the cursors used in the application program are ambiguous. The ambiguous cursors can be escalated to isolation level CS because of a BLOCKING option. The default for the BLOCKING option is UNAMBIG. This means that ambiguous cursors are treated as updatable and the escalation of the isolation level to CS occurs. To prevent this escalation, you have the following two choices:

- Modify the cursors in the application program so that they are unambiguous. Change the SELECT statements to include the FOR READ ONLY clause.
- Leave cursors ambiguous in the application program, but precompile the program or bind it with the BLOCKING ALL option to allow any ambiguous cursors to be treated as read-only when the program is run.

As in the example given for Repeatable Read, of scanning 10,000 rows, if you use Uncommitted Read, you do not acquire any row locks.

With Uncommitted Read, both non-repeatable read behavior and the phantom read phenomenon are possible. The Uncommitted Read isolation level is most commonly used for queries on read-only tables,
or if you are executing only select statements and you do not care whether you see uncommitted data from other applications.
Application alternatives

Program Preparation Steps
Application performance

**DB2 optimizer**

*Administration Guide: Performance* – Chapter 3

**Query optimization class**

- Database Configuration Parameter
  
  db2 UPDATE DB CFG FOR <database> USING DFT_QUERYOPT n

- command
  
  db2 SET CURRENT QUERY OPTIMIZATION = n

- Prep or Bind option
  
  db2 PREP pgm1.sqc QUEROPT n

- db2cli.ini file for CLI clients
  
  DB2 OPTIMIZATION n
**Rebinding applications**

*Application Development Guide: Programming Client Applications*

**Implicit rebinding**

Implicit rebinding is automatic at next execution and triggered if:

- Drop object referenced in package
- Drop primary key on table referenced in package
- Drop/add referential constraint on parent or child table referenced in package
- Revoke **privilege required by binder** to execute static SQL statement embedded in package

```
SYSCAT.PACKAGES
PKGSHEMA | PKGNAME   | BOUNDBY | VALID | ...
```

- **Y** → can run without rebinding
- **N** → **cannot run without rebinding**
- **X** → package is inoperative

**authorization id of package binder**

**Explicit rebinding**

```
REBIND [PACKAGE] package-name
```

- **NEW INDEXES AVAILABLE**
- **STATISTICS IN CATALOG HAVE CHANGED**
- **CONTROL WHEN INVALID PACKAGES ARE BOUND**
- **MUST EXPLICITLY BIND INOPERATIVE PACKAGES**
Configuration parameters

Concurrent application tuning
Administration Guide: Performance – Chapter 8

Asynchronous page cleaner
Administration Guide: Performance – Chapter 13
**Blocking**

```
DECLARE AMBIG CURSOR
FOR SELECT NAME, STYLE
FROM ARTISTS
.

DECLARE UPDTABL CURSOR
FOR SELECT EARNINGS, NAME
FROM ARTISTS
FOR UPDATE OF EARNINGS
.

DECLARE ROVER CURSOR
FOR SELECT NAME, LABEL
FROM ARTISTS
FOR FETCH ONLY
.

(DYNAMIC SQL BLOCK)
.
```

### CURSOR NAME

<table>
<thead>
<tr>
<th>BIND OPTION</th>
<th>AMBIGY</th>
<th>UPDTABL</th>
<th>ROVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>![Diagonal line]</td>
<td>![Diagonal line]</td>
<td>![Diagonal line]</td>
</tr>
<tr>
<td>UNAMBIG</td>
<td>![Diagonal line]</td>
<td>![Diagonal line]</td>
<td>![Diagonal line]</td>
</tr>
<tr>
<td>NO</td>
<td>![Diagonal line]</td>
<td>![Diagonal line]</td>
<td>![Diagonal line]</td>
</tr>
</tbody>
</table>

---

**Operational utilities**

[Command Reference Manual] – Chapter 3
Update statistics using RUNSTATS

```
RUNSTATS ON TABLE table name
  | Table Object Options
  | ALLOW WRITE ACCESS
  | ALLOW READ ACCESS

Table Object Options:
  FOR index-clause
  Column Stats Clause AND Index-Clause

Index Clause:
  INDEXES | INDEX | ALL
  | INDEXES | INDEX | ALL
  | INDEXES | INDEX | ALL
  | INDEXES | INDEX | ALL
  | INDEXES | INDEX | ALL
  | INDEXES | INDEX | ALL
  | INDEXES | INDEX | ALL

Column Stats Clause:
  ON Cols Clause
  Distribution Clause

Distribution Clause:
  WITH DISTRIBUTION
  | On Dist Cols Clause
  | Default Dist Options

On Cols Clause:
  ON ALL COLUMNS
  | ON COLUMNS
  | ON KEY COLUMNS

On Dist Cols Clause:
  ON ALL COLUMNS
  | ON COLUMNS
  | ON KEY COLUMNS

Default Dist Option:
  DEFAULT

Frequency Option
  NUM_FREQVALUES integer

Quantile Option
  NUM_QUANTILES integer

Column Option
  LIKE STATISTICS
  column name
```

Determine if table reorganization required – REORGCHK

```
REORGCHK
  UPDATE STATISTICS
  CURRENT STATISTICS
  ON TABLE USER
  ON SCHEMA schema-name
  USER TABLE SYSTEM ALL
  table-name
```
**Resulting statistics**

### Table Statistics

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>NAME</th>
<th>CARD</th>
<th>OI</th>
<th>NP</th>
<th>FP</th>
<th>TSIZE</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>REORG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMIN</td>
<td>ACCT</td>
<td>1000000</td>
<td>0</td>
<td>27792</td>
<td>27792</td>
<td>1.10e+07</td>
<td>0</td>
<td>97</td>
<td>100</td>
<td>---</td>
</tr>
</tbody>
</table>

### INDEX Statistics

<table>
<thead>
<tr>
<th>SCHEMA</th>
<th>NAME</th>
<th>LEAF</th>
<th>ELEAF</th>
<th>LVELS</th>
<th>ISIZE</th>
<th>DEL</th>
<th>KEYS</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>REORG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table: ADMIN.ACCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADMIN ACCT_GRP</td>
<td>le+006</td>
<td>170</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1009</td>
<td>4.71</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>ADMIN ACCTINK</td>
<td>le+006</td>
<td>3611</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1e+06</td>
<td>100</td>
<td>97</td>
<td>6</td>
<td>0</td>
<td>---</td>
<td></td>
</tr>
</tbody>
</table>

**Analysis**

### Table Statistics:

- **F1**: 100 * Overflow / Card < 5%
- **F2**: 100 * Table Size / ((FPages - 1) * (TABLEPAGESIZE - 75)) > 63%
- **F3**: 100 * NPages / FPages > 80%

### Index Statistics:

- **F4**: Cluster Ratio or normalized Cluster Factor > 80%
- **F5**: 100 * (KEYS * (ISIZE + 9) + (CARD - KEYS) * 5) / ((INLEAF - NUM_EMPTY_LEAFS) * INDEXPAGESIZE) > 50%
- **F6**: (100 - PCTFREE) * (INDEXPAGESIZE - 96) / (SIZE + 12) * (INLEVELS - 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
Reorganizing with REORG

```
REORG TABLE table-name Table Clause
INDEXES ALL FOR TABLE table-name Index Clause
```

Table Clause:

```
INDEX index-name
ALLOW READ ACCESS
ALLOW NO ACCESS USE tbspace INDEXSCAN LONGLOBDATA START
INPLACE ALLOW WRITE ACCESS NOTRUNCATE TABLE RESUME
STOP PAUSE
```

Index Clause:

```
ALLOW READ ACCESS
ALLOW NO WRITE ACCESS
CLEANUP ONLY ALL PAGES
```

Database Partition Clause:

```
ON ( dp:partition-num1 to dp:partition-num2 )
EXCEPT ( dp:partition-num1 to dp:partition-num2 )
```

Statistical catalog views

SYSSTAT.
- COLUMNS
- INDEXES
- COLDIST
- TABLES
- ROUTINES

Security

Administration Guide: Planning – Chapter 1
Administration Guide: Implementation – Chapter 4
**DB2 security overview**

DB2 uses a combination of:
- External security service
- Internal access control information

**Authentication**
- Identify the user
  - Check entered username and password
- Done by security facility outside of DB2

**Authorization**
- Check if authenticated user may perform requested operation
- Done by DB2 facilities
  - Information stored in DB2 catalog, DBM configuration file

**Authority levels**

**Access control authority**

Authorities provide a way both to group privileges, and to control maintenance and utility operations for instances, databases, and database objects. Users can have administrative authorities that give full privileges on a set of database objects, or they can have system authorities that give full privileges on managing the system but do not allow access to the data.

**SYSADM** is the highest level of administrative authority. It includes all the privileges on all databases within the DB2 instance, as well as the authority to grant and revoke all the other authorities and privileges.

**DBADM** provides administrative authority for a specific database. It allows the user to access and modify all the objects within that database. A user with **DBADM** authority can grant and revoke privileges on the database to others, but cannot grant or revoke **DBADM** authority.

**SYSCTRL** is the authority for controlling the resources used by the database manager (for example, creating and deleting databases), but does not allow access to the data within the databases.
SYSMAINT is the authority for performing maintenance operations, such as starting and stopping the DB2 server and backing up and restoring databases. It does not allow access to the data within the databases.

LOAD authority at the database level, combined with INSERT privilege on a table, allows the user to load data into that table.

Database-specific authorities are stored in the database catalogs; system authorities are stored in the database manager configuration file for the instance.

Database authority summary

<table>
<thead>
<tr>
<th>Function</th>
<th>SYSADM</th>
<th>SYSCTRL</th>
<th>SYSMAINT</th>
<th>DBADM</th>
<th>LOAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIGRATE DATABASE</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE DBM CFG</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRANT/REVOKE DBADM</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE dbwhd/cdsc directories</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORCE USERS OFF SYSTEM</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE/DROP DATABASE</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE/DROP/ALTER TABLE SPACE</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTORE TO NEW DATABASE</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATE DD CFG</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BACKUP DATABASE or TABLE SPACE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTORE TO EXISTING DATABASE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFORM ROLLFORWARD RECOVERY</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>START/STOP DATABASE INSTANCE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESTORE TABLE SPACE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUN TRACE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAKE DBM or DB SNAPSHOTS</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>QUERY TABLE SPACE STATE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE LOG HISTORY FILES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>QUESCE TABLE SPACE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>RERORG TABLE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>RUNRUNSTATS UTILITY</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>READ LOG FILES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>CREATE/ACTIVATE/DROP EVENT MONITORS</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td>YES</td>
</tr>
</tbody>
</table>
**Privileges**

**Authorities and privileges**

A privilege conveys the right for an authorized user, by means of their authorization ID, to perform a specific action on a specific object. It enables that authorization ID to create, modify, or access a certain database resource. An authorization ID may represent a user, a group, or all users (PUBLIC).

Privileges enable a user to create or access database resources. Authorities provide a way both to group privileges, and to control maintenance and utility operations for instances, databases, and database objects.

Privileges can be granted explicitly using the GRANT SQL statement or implicitly when executing a package or another privilege. For example, when a user executes a package that involves other privileges they obtain those privileges.

- **CONNECT** allows a user to access a database.
- **CREATEIN** allows a user to create objects within a schema.
- **USE** allows a user to create tables in a table space.
- **INDEX** allows a user to create indexes on a table.
- **DELETE** allows a user to delete rows from a table or view.
- **EXECUTE** allows a user to execute a package.
- **CONTROL** is like a master privilege on some database objects.
- **WITH GRANT** option on GRANT allows a user to grant the privilege to others.

Users with SYSADM or DBADM authority or CONTROL privilege can explicitly grant and revoke privileges using the GRANT and REVOKE SQL statements.

Privileges can also be granted implicitly, that is when a user is explicitly granted certain higher-level privileges. Implicit privileges can also be granted to a user who has the privilege to execute a package. For example, when a user executes a package that involves other privileges they obtain those privileges.
only while executing that package. They do not necessarily require the explicit privileges to work directly with the data objects used by the package.

Groups can be used to provide authorization for a collection of users without having to grant or revoke privileges for each user individually. However, group privileges cannot be used in static SQL or the creation of objects such as triggers and views, except for the predefined group called PUBLIC.

**Controlling use of schemas**

- Names collection of objects
- Forms high-order part of objects with a two part names, for example MEL.T1
- User with DBADM authority creates schema PAY for user MEL
  - CREATE SCHEMA AUTHORIZATION MEL
- Mel can create objects in schema pay
  - CREATE TABLE PAY.T1 (COL1 INT)
- Mel can grant privileges to other users:
  - GRANT CREATEIN ON SCHEMA PAY TO USER CAL
  - GRANT ALTERIN, CREATEIN, DROPIN ON SCHEMA PAY TO GROUP G1 WITH GRANT OPTION
- Achieving greater schema control:
  - REVOKE IMPLICIT_SCHEMA ON DATABASE FROM PUBLIC
  - GRANT IMPLICIT_SCHEMA ON DATABASE TO USER JON

**Protecting resources through programs**
Explicit privileges

GRANT TABLE/VIEW privileges support

User DB2A is a DBADM

GRANT SELECT, INSERT ON PAY.T2 TO JON WITH GRANT OPTION

SELECT * FROM SYSCAT.TABAUTH WHERE GRANTEE = 'JON'

<table>
<thead>
<tr>
<th>Grantor</th>
<th>Grantee</th>
<th>Schema</th>
<th>Tablename</th>
<th>Control</th>
<th>Alter</th>
<th>Delete</th>
<th>Index</th>
<th>Insert</th>
<th>Select</th>
<th>Ref</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2A</td>
<td>JON</td>
<td>PAY</td>
<td>T2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>G</td>
<td>G</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
User DB2A is a DBADM

GRANT SELECT, INSERT ON PAY.T2 TO JON WITH GRANT OPTION

SELECT * FROM SYSCAT.TABAUTH
WHERE GRANTEE = 'JON'

<table>
<thead>
<tr>
<th>Grantor</th>
<th>Grantee</th>
<th>Schema</th>
<th>Tabname</th>
<th>Control</th>
<th>Alter</th>
<th>Delete</th>
<th>Index</th>
<th>Insert</th>
<th>Select</th>
<th>Ref</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2A</td>
<td>JON</td>
<td>PAY</td>
<td>T2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>G</td>
<td>G</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

User DB2A is a DBADM

GRANT CONTROL ON PAY.T3 TO MEL

SELECT * FROM SYSCAT.TABAUTH
WHERE GRANTEE = 'MEL'

<table>
<thead>
<tr>
<th>Grantor</th>
<th>Grantee</th>
<th>Schema</th>
<th>Tabname</th>
<th>Control</th>
<th>Alter</th>
<th>Delete</th>
<th>Index</th>
<th>Insert</th>
<th>Select</th>
<th>Ref</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2A</td>
<td>MEL</td>
<td>PAY</td>
<td>T3</td>
<td>Y</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
</tbody>
</table>

Public privilege support for static SQL and views

MEL is a member of group 1

db2 grant select on table T1 to group1
db2 grant update on table T1 to user mel
db2 grant insert on table T1 to public
db2 grant select on table T2 to public

**Mel attempts bind of pgm.sqc**

exec sql insert into T1...
exec sql insert T1 set...
exec sql select... from T1...

**OK**

**OK**

**bind fails**

**Mel attempts to create a view**

creat view V2 as select C3 from T2
creat view V1 as
select C1, C8 from T1

**OK**

**create view fails**
**Implicit privileges**

**Create database**
- Internal GRANT of DBADM authority with BINDADD, CONNECT, CREATETAB, CREATE_EXTERNAL_ROUTINE, CREATE_NOT_FENCED_ROUTINE, IMPLICIT_SCHEMA, LOAD and QUIESCE_CONNECT privileges to creator (SYSADM or SYSCTRL)
- Internal GRANT of BINDADD, CREATETAB, CONNECT and IMPLICIT_SCHEMA to PUBLIC
- BIND and EXECUTE privilege on each successfully bound utility to PUBLIC
- SELECT on system catalog tables and views to PUBLIC
- USE privilege on USERSPACE1 table space to PUBLIC
- EXECUTE WITH GRANT privilege to PUBLIC on all functions in SYSFUN schema; EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema

**GRANT DBADM**
- Internal GRANT of BINDADD, CONNECT, CREATETAB, CREATE_EXTERNAL_ROUTINE, CREATE_NOT_FENCED_ROUTINE, IMPLICIT_SCHEMA, LOAD and QUIESCE_CONNECT

**Create object (table, index, package)**
- Internal GRANT of CONTROL to object creator

**Create view**
- Internal GRANT to intersection of creator’s privileges on base table(s) to view creator