

DB2 9 DBA exam 731 prep, Part 6: High availability: Backup and recovery

Sylvia Qi
Raul Chong

July 18, 2006

This tutorial discusses database backup and recovery topics. It explains the different methods of database recovery and logging, and how to use the `BACKUP`, `RESTORE`, `ROLLFORWARD`, and `RECOVER` commands. It also covers the new `DATABASE REBUILD` operation. This is the first of a two-part discussion of high availability; Part 7 covers split mirroring and high-availability disaster recovery. This is the sixth in a [series of seven tutorials](#) to help you prepare for the DB2® 9 for Linux®, UNIX®, and Windows™ Database Administration (Exam 731).

[View more content in this series](#)

Before you start

This is the sixth in a series of seven tutorials that you can use to help prepare for the DB2 9 for Linux, UNIX, and Windows Database Administration Certification Exam. This tutorial, combined with Part 7, High availability: Split mirroring and high-availability disaster recovery, covers the objectives in the section of the exam entitled "High Availability."

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 discusses database backup and recovery topics. It explains the different methods of database recovery and logging, and how to use the `BACKUP`, `RESTORE`, `ROLLFORWARD`, and `RECOVER` commands. It also covers the new `DATABASE REBUILD` operation. This is the sixth 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). The material in this tutorial primarily covers the objectives in Section 6 of the exam, "High availability." The remaining high availability topics are covered in Part 7, High availability: Split mirroring and HADR.

Objectives

After completing this tutorial, you should be able to:

- Understand the recovery methods available with DB2
- Understand the transaction logs and the different types of logs available
- Understand the types of logging methods that can be used
- Perform `BACKUP` operations
- Perform `RESTORE` operations
- Perform `ROLLFORWARD` operations
- Perform `RECOVER` operations
- Perform `DATABASE REBUILD` operations
- Understand index re-creation issues

Prerequisites

To understand the material presented in this tutorial you should be familiar with:

- The DB2 environment (database manager configuration files, database configuration files, DB2 registry variables, and so on)
- Use of the command line processor and DB2 GUI tools to invoke DB2 commands
- The different DB2 objects, such as buffer pools, tablespaces, tables, and indexes
- Basic SQL operations that can be performed against a database (`UPDATE`, `INSERT`, `DELETE`, and `SELECT` SQL statements)

You should also be familiar with the following terms:

- **Object:** Anything in a database that can be created or manipulated with SQL (for example, tables, views, indexes, packages).
- **Table:** A logical structure that is used to present data as a collection of unordered rows with a fixed number of columns. Each column contains a set of values, each value of the same data type (or a subtype of the column's data type); the definitions of the columns make up the table structure, and the rows contain the actual table data.
- **Record:** The storage representation of a row in a table.
- **Field:** The storage representation of a column in a table.
- **Value:** A specific data item that can be found at each intersection of a row and column in a database table.
- **Structured Query Language (SQL):** A standardized language used to define objects and manipulate data in a relational database. (For more on SQL, see the fourth tutorial in this series.
- **DB2 optimizer:** A component of the SQL precompiler that chooses an access plan for a Data Manipulation Language (DML) SQL statement by modeling the execution cost of several alternative access plans and choosing the one with the minimal estimated cost.

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.

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.

Database recovery concepts

Recovery scenarios

You never know when a disaster or failure may hit your system. It is best to be prepared and protect your data not only from external factors, but also from internal users who might inadvertently corrupt your database with incorrect information.

Do you back up your database? Will you be able to recover all the transactions being performed up to the last second?

To minimize the loss of your data, you need to have a recovery strategy, make sure it works, and constantly practice it. Some recovery scenarios you should consider are:

System outage

A power failure, hardware failure, or software failure can cause your database to be in an inconsistent state.

Transaction failure

Users can inadvertently corrupt your database by modifying it with incorrect data.

Media failure

If your disk drive becomes unusable, you may lose all or part of your data.

Disaster

The facility where your system is located might be damaged by fire, flooding, or other similar disasters.

Recovery strategies

To plan your recovery strategy, you should ask yourself some questions:

- Can your data be loaded again from another source?
- How much data can you afford to lose?
- How much time can you spend recovering the database?
- What storage resources are available for storing backups and log files?

Transactions

A *unit of work* (UOW), also called a *transaction*, consists of one or more SQL statements that end with a `COMMIT` or `ROLLBACK` statement. All of the statements inside this UOW are treated as a unit, which ensures data consistency. For example, a customer is trying to transfer \$100 from a savings account to a checking account. The UOW in this case would look like:

```
DELETE 100 dollars from SAVINGS account  
INSERT 100 dollars to CHECKING account  
COMMIT
```

If these statements were not treated as a unit, you can imagine what would happen if there were a hardware failure after the `DELETE` but before the `INSERT` statement: The customer would lose \$100! Because the statements are treated as a unit, however, this will never happen. DB2 will know the unit did not complete (`COMMIT`), and thus it will `ROLLBACK` all the changes made by prior statements and return the affected rows to the state they had prior to the beginning of the transaction.

There is no statement used to identify the beginning of a transaction. The statement following a `COMMIT` or `ROLLBACK` would start a new transaction.

Types of recovery

Let's familiarize ourselves with the types of recovery concepts. DB2 allows for the following types of recovery:

Crash recovery

Protects a database from being left in an inconsistent state by undoing (rolling back) transactions that were not committed. Consider again the example in the previous panel. If there had been a power failure prior to the `COMMIT` statement, the next time DB2 is restarted and the database accessed, DB2 would `ROLLBACK` first the `INSERT` statement and then the `DELETE` statement. (The order in which statements are rolled back is the reverse of the order in which they were originally executed.)

Version recovery

Allows for the restoration of a previous version of a database using a backup image obtained from a `BACKUP` command. The database that is restored will contain the information at the state it had when the `BACKUP` command was executed. If further activity was performed against the database after this backup was taken, this information is lost.

Rollforward recovery

Extends the version recovery by using full database backups in conjunction with log files. A backup has to be restored first to be used as a baseline; then logs are applied on top of this backup. This procedure will allow for the restoration of a database or tablespace to a particular point in time. Rollforward recovery requires *archival logging* to be enabled. Archival logging is discussed in a [Types of logging](#).

DB2 logs

DB2 transaction logs are critical for recovery. They keep track of changes made to database objects and data. Logs can be stored in files or in raw devices. For the examples below, we'll use files. To ensure data integrity, DB2 uses a write-ahead logging scheme, in which it writes to the logs before writing (externalizing) the changes to the database also on disk. The figure below shows this scheme.

In this figure there are four SQL statements that have been performed. The statements have been cached in the package cache, and the data pages have been retrieved from the database into the buffer pool. As the SQL statements are performed, the changes are first recorded in the log buffer

and then written to the log files. In this example, the new versions of the data pages have not yet been externalized to the database. This is normally done when buffer pool space is needed or performed asynchronously for performance reasons.

Primary and secondary log files

Primary log files are immediately allocated on the first database connection or at database activation time. *Secondary log files* are allocated dynamically one at a time when needed.

There are several database configuration parameters related to logging. Some of them are:

Parameter	Use
LOGPRIMARY	Indicates the number of primary log files to be allocated
LOGSECOND	Indicates the maximum number of secondary log files that can be allocated
LOGFILSIZ	Is used to specify the size of a log file (in number of 4 KB pages)

Let's consider an example. Imagine you have the following values in your database configuration file:

```
Log file size (4 KB)          (LOGFILSIZ) = 1000
Number of primary log files   (LOGPRIMARY) = 3
Number of secondary log files (LOGSECOND) = 2
Path to log files             = C:\mylogs\
```

As soon as the first connection to the database is established, three primary log files, each consisting of 1000 4 KB pages, are allocated. If you look in the C:\mylogs directory, you will see the three files:

```
Directory of C:\MYLOGS\

06/04/2006  06:17 PM          4,104,192  S0000000.LOG
06/04/2006  06:17 PM          4,104,192  S0000001.LOG
06/04/2006  06:17 PM          4,104,192  S0000002.LOG
               3 File(s)      12,312,576 bytes
```

Now, let's say there is no activity in your database, and you decide to perform the following transaction, which inserts a million records:

```
INSERT INTO TABLE1 VALUES(1);
INSERT INTO TABLE1 VALUES(2);
...
INSERT INTO TABLE1 VALUES(1,000,000);
COMMIT;
```

We mentioned that changes to the database are recorded in the logs. Without adding the complexity of calculating exactly how much space each of these inserts would take, you should get the idea of what we're trying to illustrate. DB2 will fill up the first log, and will continue with the second, and then the third. After it finishes with the third log file, there are no more primary (pre-allocated) log files, so DB2 will dynamically allocate the first secondary log file since LOGSECOND is greater than zero. Once this has filled up, DB2 will continue allocating another secondary log file

and will repeat this process for a maximum of LOGSECOND log files. For this example, when DB2 tries to allocate the third secondary log file, it will return an error indicating that a transaction full condition has been reached. At this point, the transaction will be rolled back.

Types of logs

This section briefly defines the different types of logs. In the next section, you'll see how they are used when we talk about types of logging. There are three types or states of DB2 transaction logs:

Active logs

A log is considered *active* if either of the following two conditions are satisfied:

- It contains information about transactions that have not yet been committed or rolled back.
- It contains information about transactions that have committed but whose changes have not yet been written to the database disk (externalized).

Online archive logs

Contain information for committed *and* externalized transactions. Such logs are kept in the same directory as the active logs.

Offline archive logs

Archive logs that have been moved from the active log directory to another directory or media. This move can be done either manually or automatically by DB2.

Types of logging

There are two types of logging: circular logging and archival.

Circular logging

Circular logging is the default logging mode for DB2. As the name implies, this type of logging reuses the logs in a circular mode. For example, if you had four primary logs, DB2 would use them in this order: Log #1, Log #2, Log #3, Log #4, Log #1, Log #2, and so on.

A log can be reused in circular logging as long as it only contains information about transactions that have already been committed *and* externalized to the database disk. In other words, if the log is still an active log, it cannot be reused.

Using the circular logging example, what happens if you have a long-running transaction that spans five logs? In this case, DB2 allocates another log file -- a secondary log file, as described in a [Primary and secondary log files](#). The figure below shows how this works.

Archival logging

When you use archival logging, you will be archiving (retaining) the logs. While in circular logging you would overwrite transactions that were committed and externalized, with archival logging you will keep them. For example, if you had four primary logs, DB2 may use them in this order: Log #1, Log #2, Log #3, Log #4, (archive Log #1 if all its transactions are committed and externalized), Log #5, (archive Log #2 if all its transactions are committed and externalized), Log #6, and so on.

As shown in this example, DB2 will keep four primary log files available, and will not reuse the log files that have been filled up with transactions that had already been committed and externalized.

It will not overwrite the logs that have become archive logs. The figure below illustrates how this works.

The type of logging for a database is determined by the database parameter `LOGARCHMETH1`. When `LOGARCHMETH1` is OFF (default), archival logging is disabled, and circular logging is used.

To enable archival logging, `LOGARCHMETH1` can be set to any of the following values:

Value	Description
LOGRETAIN	The log files will be retained in the active log directory.
USEREXIT	The archive and retrieval of the logs are performed automatically by a user-supplied userexit program that must be called db2uext2. This program is invoked to move online archive logs to a directory other than the active log directory, or to another media. It is also invoked to retrieve offline archive logs to the active log directory when they are needed during a ROLLFORWARD operation. db2uext2 must be stored in the sqllib\bin directory on Windows, and sqllib/adm on UNIX
DISK:directory_name	The same algorithm is used as in USEREXIT. DB2 does not call a userexit program, but it automatically archives the logs from the active log directory to the specified directory.
TSM:[management class name]	The same algorithm is used as in USEREXIT. The logs are archived on the local Tivoli Storage Manager (TSM) server. The management class name parameter is optional. If not specified, the default management class is used.
VENDOR:library_name	The same algorithm is used as in USEREXIT. The logs are archived using the specified vendor library.

For backward compatibility reasons, the database configuration file still includes the parameters `LOGRETAIN` and `USEREXIT`. These parameters have been replaced by `LOGARCHMETH1` since version 8.2. If you update the `USEREXIT` or `LOGRETAIN` parameters, `LOGARCHMETH1` will automatically be updated, and vice versa.

Infinite logging

With circular logging and archival logging, log space can potentially be filled up with active logs. With infinite logging enabled, DB2 archives a log as soon as it becomes full. It does not wait for all the transactions in the log to be committed and externalized before archiving it; this guarantees that the active log directory will never fill up. For example, if you have a long-running transaction, you will not run out of log space.

Infinite logging is not recommended, however, since it can prolong crash recovery times as active logs may need to be retrieved from the archive site. Infinite logging is a spin-off of archival logging. To enable infinite logging:

1. Set the `LOGSECOND` database configuration parameter to **-1**.
2. Enable archive logging.

Other recovery topics

Logging types and recovery types

Now that you understand the different types of logging and recovery, it is important to note that not all logging types support all recovery types. Circular logging supports only crash and version recovery, while archival logging supports all types of recovery: crash, version, and rollforward recovery.

Recoverable and nonrecoverable databases

Recoverable databases are databases that can be recovered using crash, version, or rollforward recovery; thus, archival logging needs to be enabled for these databases. *Nonrecoverable* databases are those that do not support rollforward recovery; thus, only circular logging is used.

Review

We've covered several concepts about database logs and logging so far. The following figure summarizes some of the concepts.

This figure shows several transactions running over a period of time. Some transactions run concurrently; they start filling up the log buffer first, and are subsequently written to the log files on disk.

MINCOMMIT is a database configuration file parameter.

Contents from the log buffer will be written to the log files when the log buffer is full, or when a `MINCOMMIT` number of commits are issued. Changes to the data pages for committed transactions will be externalized (written from the buffer pool to the database disk asynchronously). For simplicity's sake, in the figure we show this happening at commit time, but this is not normally the case.

The hexagon with the *active* label represents the amount of time log file X is still considered an active log. As you can see, this hexagon is on top of the squares representing part of transactions D and C in log file Y. Why is log file X still considered active even after it has filled up? Because it contains transactions that have not yet been committed and externalized. Log file X contains transactions A, B, and C. Only transactions A and B have been committed (and, for this example, externalized immediately); transaction C is still running and is also written in log file Y. When transaction C is committed in log file Y (and, for this example, externalized immediately), then log file X will no longer be considered an active log, but will become an online archive log.

Database and tablespace backup

Online vs. offline access

If we are performing an *online* operation (backup, restore, rollforward), we are allowing other users to access the database object we are working with at the same time.

If we are performing an *offline* operation, we are *not* allowing other users any access to the database object we are working on at that time.

We'll use the terms *online* and *offline* quite often in this section.

Database backup

A *database backup* is a complete copy of your database. Besides the data, a backup copy contains information about the tablespaces, containers, database configuration, log control file, and recovery history file. Note that a backup will not store the database manager configuration file or the registry variables. Only the database configuration file will be backed up.

To perform a backup, SYSADM, SYSCTRL, or SYSMANT authority is required.

Below is the syntax of the `BACKUP` command utility for this type of backup:

```
BACKUP DATABASE database-alias [USER username [USING password]]
[TABLESPACE (tblspace-name [ {,tblspace-name} ... ])] [ONLINE]
[INCREMENTAL [DELTA]] [USE {TSM | XBSA}] [OPEN num-sess SESSIONS]
[OPTIONS {options-string | options-filename}] | TO dir/dev
[ {,dir/dev} ... ] | LOAD lib-name [OPEN num-sess SESSIONS]
[OPTIONS {options-string | options-filename}]]
[WITH num-buff BUFFERS] [BUFFER buffer-size] [PARALLELISM n]
[COMPRESS [COMPRLIB lib-name [EXCLUDE]] [COMPROPTS options-string]]
[UTIL_IMPACT_PRIORITY [priority]] [{INCLUDE | EXCLUDE} LOGS] [WITHOUT PROMPTING]
```

Let's look at some examples to see how some of these options work.

To perform a full offline backup of the database "sample" and store the backup copy to the directory d:\mybackups, use the following command:

```
BACKUP DATABASE sample
TO d:\mybackups
```

To perform a full offline backup of the database sample using other backup options, you can use the following command:

```
(1)  BACKUP DATABASE sample
(2)  TO /db2backup/dir1, /db2backup/dir2
(3)  WITH 4 BUFFERS
(4)  BUFFER 4096
(5)  PARALLELISM 2
```

Let's look at the command above in more detail:

1. Indicates the name (or alias) of the database to back up.
2. Specifies the location(s) where you want to store the backup.
3. Indicates how many buffers from memory can be used during the backup operation. Using more than one buffer can improve performance.
4. Indicates the size of each buffer.
5. Determines how many media readers/writer processes/threads are used to take the backup.

With version 9, it is recommended to work with default values for most options, as the backup utility can determine itself what is best for its own performance.

There is no keyword `OFFLINE` in the syntax, as this is the default mode. To perform a full *online* backup of the sample database, you must specify the keyword `ONLINE`. Online backups require archive logging to be enabled for the database. The example below shows how to issue an online backup.

```
BACKUP DATABASE sample  
ONLINE  
TO /dev/rdir1, /dev/rdir2
```

Because an online backup lets users access the database while it is being backed up, changes made by these users are not likely stored in the backup copy. Therefore, an online backup is not enough for recovery, and the corresponding logs collected during the backup operation are also needed. DB2 makes it easy to collect the current active log when a backup finishes by forcing it to close (and be archived) when the online backup finishes.

To back up the logs as part of the backup copy, use the `INCLUDE LOG` option of the `BACKUP DATABASE` command. This ensures that even if you lose your logs, you can still restore to a minimum point in time using the logs that are included in the backup image. The `INCLUDE LOGS` option applies to online backups only.

For example, to take an online backup of the sample database along with the logs using the destination directory `/dev/rdir1`, issue:

```
BACKUP DATABASE sample  
ONLINE  
TO /dev/rdir1 INCLUDE LOGS
```

Tablespace backup

In a database where only some of your tablespaces change considerably, you may opt not to back up the entire database, but only specific tablespaces.

To perform a tablespace backup, use the following syntax:

```
(1) BACKUP DATABASE sample  
(2) TABLESPACE ( syscatspace, userspace1, userspace2 )  
(3) ONLINE  
(4) TO /db2tbsp/backup1, /db2tbsp/backup2
```

Line 2 in the above example indicates that this will be a tablespace backup as opposed to a full database backup. Also notice that you can include as many tablespaces in the backup as you'd like. Temporary tablespaces cannot be backed up using a tablespace level backup.

You would typically want to back up related tablespaces together. For example, imagine you're using DMS tablespaces, where one tablespace is used for the table data, another for the indexes, and another for the LOBs. You should back up all of these tablespaces at the same time so that you have consistent information. This is also true for tablespaces containing tables defined with referential constraints between them.

Incremental backups

There are two kinds of incremental backups:

Incremental

DB2 backs up all of the data that has changed since the last full database backup.

Delta

DB2 backs up only the data that has changed since the last successful full, incremental, or delta backup.

The following figure shows the differences between these types.

In the upper part of the figure, if there were a crash after the incremental backup on Friday, you could restore the first Sunday full backup, followed by the incremental backup taken on Friday.

In the lower part of the figure, if there were a crash after the delta backup on Friday, you could restore the first Sunday full backup followed by each of the delta backups taken from Monday until Friday inclusive.

Performing backups with the Control Center

The figure below shows you how to invoke the `BACKUP` utility from the Control Center. To perform a database or tablespace backup, right-click the database you want to back up and select **Backup**.

The next figure shows the Backup Wizard and the options you need to complete to execute the `BACKUP` utility. We encourage you to try this out on your own.

Naming convention for backup files

The naming convention for DB2 backup files contains the following items:

- Database alias
- Digit indicating the type of backup (0 for a full database, 3 for a tablespace backup, 4 for a copy from `LOAD`)
- Instance name
- Database node (always `NODE0000` for a single-partition database)
- Catalog node number (always `CATN0000` for a single-partition database)
- Timestamp of the backup
- Image sequence number

The following figure shows the naming convention, which applies to all platforms.

Database and tablespace recovery

Database recovery

This section discusses the `RESTORE` utility, which uses a backup file as input and a new or existing database as output. Though we are discussing database *recovery*, the utility we'll use is called `RESTORE`, not `RECOVER`.

`SYSADM`, `SYSCTRL`, or `SYSMAINT` authority is required to restore to an existing database. `SYSADM` or `SYSCTRL` authority is required to restore to a new database.

Here's the syntax of the `RESTORE` command:

```
RESTORE DATABASE source-database-alias { restore-options | CONTINUE | ABORT }
restore-options:
[USER username [USING password]]
[REBUILD WITH [ALL TABLESPACES IN DATABASE | ALL TABLESPACES IN IMAGE]
 [EXCEPT rebuild-tablespace-clause] [rebuild-tablespace-clause]
 [{TABLESPACE [ONLINE] | TABLESPACE (tblspace-name [ {,tblspace-name} .. }) [ONLINE] |
 HISTORY FILE [ONLINE]]} [INCREMENTAL [AUTOMATIC | ABORT]]
 [{USE {TSM | XBSA} [OPEN num-sess SESSIONS] |
 FROM dir/dev [ {,dir/dev} ... ] | LOAD shared-lib
 [OPEN num-sess SESSIONS]]} [TAKEN AT date-time] [TO target-directory]
 [INTO target-database-alias] [NEWLOGPATH directory] [LOGS FROM directory]
 [LOGTARGET directory]
 [WITH num-buff BUFFERS] [BUFFER buffer-size]
 [DLREPORT file-name] [REPLACE EXISTING] [REDIRECT] [PARALLELISM n]
 [WITHOUT ROLLING FORWARD] [WITHOUT DATALINK] [WITHOUT PROMPTING]
Rebuild-tablespace-clause:
[TABLESPACE (tblspace-name [ {,tblspace-name} ... ])]
```

Let's look at an example. To perform a restore of the sample database, use the following command:

```
(1)RESTORE DATABASE sample
(2) FROM C:\DBBACKUP
(3) TAKEN AT 20060314131259
(4) WITHOUT ROLLING FORWARD
(5) WITHOUT PROMPTING
```

In the example above,

1. Indicates the name of the database image to restore.
2. Specifies the location from which the input backup file is to be read.
3. Should there be more than one backup image in the directory, this option would identify the specific backup based on the timestamp, which is part of the backup name.
4. If a database had archival logging enabled, it is automatically placed in rollforward pending state when it is restored. This line tells DB2 not to place the database in rollforward pending state.
5. You will not be prompted while the `RESTORE` is being performed.

Note there is no keyword `OFFLINE` in the syntax, as this is the default mode. For the `RESTORE` utility, this is the only mode allowed for databases.

If the logs are included in the backup image, you can restore the logs files using the `LOGTARGET` option of the `RESTORE DATABASE` command.

To restore the `SAMPLE` database from a backup image residing in the `C:\DBBACKUP` directory, and restore the log files to `C:\DB2\NODE0000\SQL00001\SQLOGDIR` directory, issue:

```
RESTORE DATABASE sample
FROM C:\DBBACKUP
LOGTARGET C:\DB2\NODE0000\SQL00001\SQLOGDIR
```

You can also restore only the log files without restoring the database using the `LOGS` keyword:

```
RESTORE DATABASE sample  
LOGS FROM C:\DBBACKUP  
LOGTARGET C:\DB2\NODE0000\SQL00001\SQLLOGDIR
```

Tablespace recovery

You can restore tablespaces either from a full database backup or from a tablespace backup. Tablespace recovery requires some careful planning, as it is easier to make mistakes that would put your data into an inconsistent state.

Here is an example of a tablespace `RESTORE` command:

```
(1) RESTORE DATABASE sample  
(2) TABLESPACE ( mytblspace1 )  
(3) ONLINE  
(4) FROM /db2tbsp/backup1, /db2tbsp/backup2
```

In the example above,

1. Indicates the name of the database image to restore.
2. Indicates that this is a tablespace `RESTORE`, and specifies the name of the tablespace(s) to restore.
3. Indicates that this is an online restore. For user tablespaces, both online and offline restores are allowed. As mentioned earlier, for databases, only offline restores are allowed.
4. Specifies the location where the input backup file is located.

Tablespace recovery considerations

After a tablespace is restored, it will *always* be placed in rollforward pending state. To make the tablespace accessible and reset this state, the tablespace must be rolled forward at least to a minimum point in time. This minimum point in time ensures that the tablespace and logs are consistent with what is in the system catalogs.

Consider this example:

1. Imagine that at time t1 you took a full database backup, which included the tablespace mytbls1.
2. At time t2, you created the table myTable in the tablespace mytbls1. This would set the minimum point in time for recovery of the tablespace mytbls1 to t2.
3. At time t3, you decided to restore only tablespace mytbls1 from the full database backup taken at t1.
4. After the restore is complete, tablespace mytbls1 will be placed in rollforward pending state. If you were allowed to roll forward to a point prior to the minimum point in time, tablespace mytbls1 will not have the table myTable; however, the system catalog would say that the table does exist in mytbls1. To avoid inconsistencies like this, DB2 will force you to roll forward at least to the minimum point in time when you restore a tablespace.

A minimum point in time is updated when DDL statements are run against the tablespace, or against tables in the tablespace. To determine the minimum point in time of recovery for a tablespace, you can use either of the following methods:

- Use the `LIST TABLESPACES SHOW DETAIL` command.
- Obtain a tablespace snapshot with the `GET SNAPSHOT FOR TABLESPACE ON db_name` command.

The system catalog tablespace (SYSCATSPACE) must also be rolled forward to the end of logs and in offline mode. We'll discuss more about the `ROLLFORWARD` command in [Database and tablespace rollforward](#).

Performing restores with the Control Center

The figure below shows you how to invoke the `RESTORE` utility from the Control Center. To perform a database or tablespace restore, right-click the database you want to restore and select **Restore**. We encourage to try this on your own.

The next figure shows some options you need to complete to execute the `RESTORE` utility. We encourage you to try this on your own.

Incremental restore

Recall that an incremental backup is a backup image that contains only pages that have been updated since the previous backup was taken. To perform a restore using incremental backups, specify the `INCREMENTAL` option on the `RESTORE DATABASE` command. Usually an incremental restore involves a series of restore operations. Each backup image involved in the restore must be restored in the following order: last, first, second, third and so on up to and including the last image.

Let's look at an example. The following is a sample weekly incremental backup strategy for a recoverable database MYDB. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

```
1. (Sun) backup db mydb use tsm
2. (Mon) backup db mydb online incremental delta use tsm
3. (Tue) backup db mydb online incremental delta use tsm
4. (Wed) backup db mydb online incremental use tsm
5. (Thu) backup db mydb online incremental delta use tsm
6. (Fri) backup db mydb online incremental delta use tsm
7. (Sat) backup db mydb online incremental use tsm
```

To restore the database using the incremental backup created on Friday, we need to issue a series of `RESTORE DATABASE` commands to restore each backup image that is needed, in this order:

```
1. restore db mydb incremental taken at (Fri)
2. restore db mydb incremental taken at (Sun)
3. restore db mydb incremental taken at (Wed)
4. restore db mydb incremental taken at (Thu)
5. restore db mydb incremental taken at (Fri)
```

As you can see, this process is very lengthy and is also error-prone. You have to know what images to restore and in what order they must be restored. To make this process easier, use the automatic incremental restore utility. To perform an automatic incremental restore, all you need to do is identify the most recent backup image you want to restore, and issue only one `RESTORE DATABASE` command against it, with the `INCREMENTAL AUTOMATIC` option. The restore utility will take care of the rest.

All the above manual restore commands can be replaced by one automatic incremental restore command:

```
restore db mydb incremental automatic taken at (Fri)
```

Redirected restore

We mentioned earlier that a backup file includes information about tablespaces and containers. What would happen if a container that used to exist when the backup was taken exists no longer? If the `RESTORE` utility cannot find this container, you will get an error.

What if you don't want to restore this backup at this location, but somewhere else where other configurations are used? Again, restoring the backup in this scenario would cause a problem.

Redirected restores solve these problems. A redirected restore simply restores the backup in four steps. It will:

1. Obtain the information about the containers and tablespaces recorded in the input backup. This is done by including the `REDIRECT` keyword as part of the `RESTORE` command. For example:

```
RESTORE DATABASE DB2CERT FROM C:\DBBACKUP
      INTO NEWDB REDIRECT WITHOUT ROLLING FORWARD
```

And here's the output from this command:

```
SQL1277W  A redirected restore operation is being performed.  Table space
configuration can now be viewed and table spaces that do not use automatic
storage can have their containers reconfigured.
DB20000I  The RESTORE DATABASE command completed successfully.
```

2. Review the tablespace information from the (partially) restored database newdb:

```
LIST TABLESPACES SHOW DETAIL
```

3. Set the new containers for each tablespace. A tablespace has an ID, which can be obtained from the output of the `LIST TABLESPACES` command. This ID is used as follows:

```
SET TABLESPACE CONTAINERS FOR 0 USING (FILE "d:\newdb\cat0.dat" 5000)
SET TABLESPACE CONTAINERS FOR 1 USING (FILE "d:\newdb\cat1.dat" 5000)
...
SET TABLESPACE CONTAINERS FOR n USING (PATH "d:\newdb2")
```

where `n` represents an ID of one of the tablespaces in the backup. Note that with redirected restores, you cannot change the type of the tablespace; if the tablespaces is SMS, it cannot be changed to DMS.

4. Start restoring the data itself into the new containers by including the keyword `CONTINUE`, as shown below:

```
RESTORE DATABASE DB2CERT CONTINUE
```

You have now seen how a redirected restore works. It can also be used to add containers for SMS tablespaces. If you reviewed the second tutorial in this series (see [Resources](#)), you should know that in most cases SMS tablespaces cannot be altered to add a container. A redirected restore provides a workaround to this limitation.

Rather than doing the above four steps manually, the redirect restore utility lets you generate a redirected restore script by issuing the `RESTORE` command with the `REDIRECT` and the `GENERATE SCRIPT` options. When the `GENERATE SCRIPT` option is used, the restore utility extracts container information from the backup image, and generates a CLP script that includes all of the detailed container information. You can then modify any of the paths or container sizes in the script, and run the CLP script to recreate the database with the new set of containers.

For example, to perform a redirected restore of the DB2CERT database using a script, follow these steps:

1. Use the restore utility to generate a redirected restore script.

```
RESTORE DATABASE DB2CERT FROM C:\DBBACKUP INTO NEWDB REDIRECT GENERATE SCRIPT
NEWDB.CLP WITHOUT ROLLING FORWARD
```

This creates a redirected restore script called *target-database-alias*.clp. In this case, newdb.clp.

newdb.clp contains all the commands required to perform a redirected restore. It also contains all the tablespace information. The newdb.clp script is shown below.

```
-- *****
-- ** automatically created redirect restore script
-- *****
UPDATE COMMAND OPTIONS USING S ON Z ON NEWDB_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM 0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
-- *****
-- ** automatically created redirect restore script
-- *****
RESTORE DATABASE DB2CERT
-- USER username
-- USING 'password'
FROM 'C:\dbbackup' TAKEN AT 20060516120102
-- ON 'D:'
-- DBPATH ON 'target-directory'
INTO NEWDB
-- NEWLOGPATH 'D:\DB2\NODE0000\SQL00001\SQLOGDIR\'
-- WITH num-buff BUFFERS
-- BUFFER buffer-size
-- REPLACE HISTORY FILE
-- REPLACE EXISTING REDIRECT
-- PARALLELISM n WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- *****
-- ** table space definition
-- *****
-- *****
-- ** Tablespace name                = SYSCATSPACE
-- ** Tablespace ID                  = 0
-- ** Tablespace Type                = Database managed space
-- ** Tablespace Content Type        = All permanent data. Regular table
--                                   space.
-- ** Tablespace Page size (bytes)   = 4096
-- ** Tablespace Extent size (pages) = 4
-- ** Using automatic storage        = Yes
-- ** Auto-resize enabled             = Yes
-- ** Total number of pages          = 16384
-- ** Number of usable pages         = 16380
-- ** High water mark (pages)        = 8872
-- *****
```



```

-- *****
-- ** Tablespace name                = TEMPSPACE1
-- ** Tablespace ID                  = 1
-- ** Tablespace Type                 = System managed space
-- ** Tablespace Content Type        = System Temporary data
-- ** Tablespace Page size (bytes)   = 4096
-- ** Tablespace Extent size (pages) = 32
-- ** Using automatic storage        = Yes
-- ** Total number of pages          = 1
-- *****
-- *****
-- ** Tablespace name                = USERSPACE1
-- ** Tablespace ID                  = 2
-- ** Tablespace Type                 = Database managed space
-- ** Tablespace Content Type        = All permanent data. Large table
--                                     space.
-- ** Tablespace Page size (bytes)   = 4096
-- ** Tablespace Extent size (pages) = 32
-- ** Using automatic storage        = Yes
-- ** Auto-resize enabled             = Yes
-- ** Total number of pages          = 8192
-- ** Number of usable pages         = 8160
-- ** High water mark (pages)        = 1888
-- *****
-- *****
-- ** Tablespace name                = TBS1
-- ** Tablespace ID                  = 5
-- ** Tablespace Type                 = Database managed space
-- ** Tablespace Content Type        = All permanent data. Large table
--                                     space.
-- ** Tablespace Page size (bytes)   = 4096
-- ** Tablespace Extent size (pages) = 32
-- ** Using automatic storage        = No
-- ** Auto-resize enabled             = No
-- ** Total number of pages          = 1000
-- ** Number of usable pages         = 960
-- ** High water mark (pages)        = 96
-- *****
SET TABLESPACE CONTAINERS FOR 5
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
FILE 'd:\DB2\DB2CERT\tbs1' 1000
);
-- *****
-- ** start redirected restore
-- *****
RESTORE DATABASE NEWDB CONTINUE;
-- *****
-- ** end of file
-- *****

```

The '--' indicates a comment. The `SET TABLESPACE CONTAINER` command is only created for tablespaces that are not set up to use automatic storage. For tablespaces that are using automatic storage, their containers are handled by DB2 automatically, thus there is no need to reset them.

2. Open the redirected restore script in a text editor to make any modifications that are required.

You can modify:

- Restore options
- Container layout and paths

3. Run the modified redirected restore script.

```
db2 -tvf newdb.clp
```

The output of the script will be written into a file called *dbname_nodenum.out*.

Database and tablespace rollforward

Database rollforward

The `ROLLFORWARD` command allows for point-in-time recovery, meaning that the command will let you traverse the DB2 logs and redo or undo the operations recorded in the log up to a specified point in time. Although you can roll forward your database or tablespace to any point in time after the minimum point in time, there is no guarantee that the end time you choose to roll forward will have all data in a consistent state.

Though the `QUIESCE` command is not discussed in this tutorial, it is worth mentioning that this command can be used during regular database operations to set consistency points. By setting these consistency points, you can always perform a point in time recovery to any of them and be assured that your data is in synch.

Consistency points, along with a lot of other information, are recorded in the DB2 history file, which can be reviewed with the `LIST HISTORY` command.

During the rollforward processing, DB2 will:

1. Look for the required log file in the current log path.
2. Reapply transactions from the log file if this log is found.
3. Search in the path specified by the `OVERFLOW LOG PATH` option and use the logs in that location if the log file is not found in the current log path.
4. Use the method specified in `LOGARCHMETH1` to retrieve the log file from the archive path if the log file is not found in the current path and the `OVERFLOW LOG PATH` option is not used.
5. Reapply the transactions, once the log is in the current log path or the `OVERFLOW LOG PATH`.

`SYSADM`, `SYSCTRL`, or `SYSMAINT` authority is required to perform the `ROLLFORWARD` command.

The syntax of the `ROLLFORWARD` command is:

```
ROLLFORWARD DATABASE database-alias [USER username [USING password]]
[TO {isotime [ON ALL DBPARTITIONNUMS] [USING LOCAL TIME | USING UTC TIME] |
END OF LOGS [On-DbPartitionNum-Clause]]] [AND {COMPLETE | STOP}] |
{COMPLETE | STOP | CANCEL | QUERY STATUS [USING LOCAL TIME | USING UTC TIME]}
[On-DbPartitionNum-Clause] [TABLESPACE ONLINE | TABLESPACE (tblspace-name
[ {,tblspace-name} ... ]) [ONLINE]] [OVERFLOW LOG PATH (log-directory
[{,log-directory ON DBPARTITIONNUM db-partition-number} ... ])] [NORETRIEVE]
[RECOVER DROPPED TABLE dropped-table-id TO export-directory]
```

```
On-DbPartitionNum-Clause:
ON [{DBPARTITIONNUM | DBPARTITIONNUMS} (db-partition-number
[TO db-partition-number] , ... ) | ALL DBPARTITIONNUMS [EXCEPT
{DBPARTITIONNUM | DBPARTITIONNUMS} (db-partition-number
[TO db-partition-number] , ... )]]]
```

Let's look at an example. To perform a rollforward of the sample database, you can use any of the following statements:

```
(1)ROLLFORWARD DATABASE sample TO END OF LOGS AND COMPLETE
(2)ROLLFORWARD DATABASE sample TO timestamp AND COMPLETE
(3)ROLLFORWARD DATABASE sample TO timestamp USING LOCAL TIME AND COMPLETE
```

From the code above:

1. In this example, we'll roll forward to the end of the logs, which means that all archived and active logs will be traversed. At the end, it will complete the rollforward and remove the rollforward-pending state by rolling back any uncommitted transactions.
2. For this example, DB2 will roll forward to the specified point in time. The timestamp used has to be in CUT (Coordinated Universal Time), which can be calculated by subtracting the local time from the current time zone.
3. This example is similar to the previous one, but the timestamp can be expressed in local time.

There is no keyword `OFFLINE` in the syntax, as this is the default mode. For the `ROLLFORWARD` command, this is the only mode allowed for databases.

Tablespace rollforward

Tablespace rollforwards can generally be either online or offline. The exception is the system catalog tablespace (SYSCATSPACE), which can only be rolled forward offline.

Here's an example tablespace rollforward:

```
ROLLFORWARD DATABASE sample
  TO END OF LOGS AND COMPLETE
  TABLESPACE ( userspace1 ) ONLINE
```

The options in this example were explained in the database rollforward section. The only new thing here is the `TABLESPACE` option, which specifies the tablespace to be rolled forward.

Tablespace rollforward considerations

If the registry variable `DB2_COLLECT_TS_REC_INFO` is enabled, only the log files required to recover the tablespace are processed. The `ROLLFORWARD` command will skip over log files that are not required, which can speed up recovery time.

The `QUERY STATUS` option of the `ROLLFORWARD` command can be used to list the:

- Log files that DB2 has rolled forward.
- Next archive log file required.
- Timestamp of the last committed transaction since rollforward processing began.

For example:

```
ROLLFORWARD DATABASE sample QUERY STATUS USING LOCAL TIME
```

After a tablespace point in time rollforward operation completes, the tablespace is put in backup-pending state. A backup of the tablespace or database must be taken because all updates made

to it between the point in time that the tablespace was recovered to and the current time have been lost.

Performing rollforwards operations with the Control Center

The figure below shows how to invoke the `ROLLFORWARD` command from the Control Center. To perform a database or tablespace rollforward, right-click the database you want to roll forward and select **Roll-forward**. We encourage you to try this on your own.

The following figure shows some options you need to complete to execute the `ROLLFORWARD` command. We encourage you to try this on your own.

Review and examples

So far we've discussed the `BACKUP`, `RESTORE`, and `ROLLFORWARD` commands. The figures below illustrate the different types of recovery that you should now understand.

For this scenario, circular logging is in effect:

At t6 there is an unscheduled power shutdown in your building. At t7, DB2 is restarted, and when you connect to the database, crash recovery is started automatically (assuming `db cfg AUTORESTART` is ON; otherwise you have to start it manually with a `RESTART DATABASE` command). Crash recovery will traverse the active logs and will redo committed transactions. If a transaction was not committed, it will be rolled back (undone). For this example, the two insert statements will be redone and the delete statement will be undone.

For this scenario, circular logging is in effect:

At t7 you realize your data in all tablespaces has been corrupted by some transaction that started at t6. At t8 you decide to restore from the full database backup taken at t1. Because circular logging is in effect, many of the committed and externalized transactions in the logs have been overwritten. Thus, logs cannot be applied (the `ROLLFORWARD` command cannot be run in circular logging, so you cannot even roll forward active logs). Conclusion: Many of the good transactions for t2 to t4 will be lost.

For this scenario, archival logging is in effect:

This is an extension of the previous scenario. In this case, the logs have been kept (archive logs); after the full database restore is applied at t8, you can rollforward the logs at t9. Logs can be rolled forward from t1 to any point in time, but likely you don't want to go past t6, when the bad transaction started.

The following scenario reviews all these concepts in more detail.

1. An offline database backup finished at t1.
2. Daily transactions are performed at t2.
3. At t4 you realize one of the transactions has corrupted tablespace Z, and you stop this transaction. Other transactions against other tablespaces continue.

4. Only for tablespace Z, you want the data at the state it was prior to t3 (prior to the start of the bad transaction), thus:

- At t5, you restore tablespace Z from the full offline backup taken at t1.
- After the restore has finished, the tablespace will be left in rollforward pending state.
- At t6 you rollforward the tablespace up to t3, prior to the start of the bad transaction.
- You have just performed a point in time recovery. Because of this, DB2 will now put the tablespace in backup pending state for consistency reasons.
- At t7 you back up the tablespace. At this point, your database is consistent and all users and applications can work normally. The restored tablespace will have a gap from t3 to t7, which is what we intended -- to remove the corrupted data.

Recover database utility

The recover database utility combines the RESTORE utility and the ROLLFORWARD utility in one easy to use command. It performs the necessary restore and rollforward operations to recover a database to a specified time, based on information found in the recovery history file. It automatically selects the best suitable backup image to perform the recovery operations.

The syntax of the `RECOVER DATABASE` command is:

```
RECOVER DATABASE source-database-alias TO isotime [USING LOCAL TIME]
[USER username [USING password]
[USING HISTORY FILE history-file]
[OVERFLOW LOG PATH directory]
[RESTART]
```

Examples

A SAMPLE database currently exists on your development server. A power failure last night has damaged the database with data corruption. You need to recover the database as soon as possible. You can do this by first locating the appropriate backup, then locating the required DB2 logs to roll forward to a point in time just before the power failure. An easier way to recover the SAMPLE database is to issue the `RECOVER DB` command, as follows:

```
(1) RECOVER DB sample
(2) RECOVER DB sample TO 2006-05-21-13.50.00 USING LOCAL TIME
```

In line 1 above, DB2 recovers the SAMPLE database from the best available backup image, and will rollforward to end of logs.

In line 2, DB2 recovers the SAMPLE database to a point in time, 2006-05-21-13.50.00, specified in local time.

Recall that the `RECOVER DATABASE` utility relies on the database recovery history file to find the best suitable backup image to restore. We did not specify the location of the history file in our recover commands above. Because the SAMPLE database already exists on the server, DB2 is able to locate the history file under the database directory path.

If the database to be recovered does not already exist, then the location of the history file must be specified.

```
RECOVER DB sample TO END OF LOGS USING HISTORY FILE  
(/home/user/oldfiles/db2rhist.asc)
```

A valid history file, containing the backup image and logs required, must exist on the server you want to recover to. In our command above, if we did not have a copy of the history file (from file transfer or history file backup) readily available, then we must somehow extract the history from the backup image itself before we can run the `RECOVER DATABASE` command. In this case, it might be easier to recover the database by running the standard `RESTORE` and `ROLL FORWARD` commands sequentially.

If, for whatever reason, a recover operation is interrupted before it successfully completes, you can restart it by rerunning the same command. If it was interrupted during the rollforward phase, then recover utility will attempt to continue the previous recover operation without redoing the restore phase. If you want to force the recover utility to redo the restore phase, issue the `RECOVER DATABASE` command with the `RESTART` option to force the recover utility to ignore any prior recover operation that failed to complete. If the recover utility was interrupted during the restore phase, then it will start from the beginning.

The recover utility does not support the following `RESTORE DATABASE` command options:

Command	Description
<code>TABLESPACE tablespace-name</code>	Table space restore operations are not supported.
<code>INCREMENTAL</code>	Incremental restore operations are not supported.
<code>OPEN num-sessions SESSIONS</code>	You cannot indicate the number of I/O sessions that are to be used with TSM or another vendor product.
<code>BUFFER buffer-size</code>	You cannot set the size of the buffer used for the restore operation.
<code>DLREPORT filename</code>	You cannot specify a file name for reporting files that become unlinked.
<code>PARALLELISM n</code>	You cannot indicate the degree of parallelism for the restore operation.
<code>WITHOUT PROMPTING</code>	You cannot specify that a restore operation is to run unattended.

Database rebuild

What is a database rebuild?

The database rebuild function is provided by the restore utility. It lets you rebuild a brand new database using a set of backup images. You can choose to rebuild the entire database, or a database with only a subset of tablespaces in the original database. The database rebuild procedure depends on whether the database is recoverable or non-recoverable. We'll discuss both scenarios in the following sections.

Rebuilding a recoverable database using tablespace backups

In the case of recoverable databases, the rebuild utility allows you to rebuild an entire database using only tablespace backups. Full database backups are no longer required. Full database backups may require larger maintenance windows, which are increasingly harder to schedule for high availability shops. The ability to rebuild a database from table space backups is a great enhancement for availability and recoverability.

Let's say you have a recoverable database called TEST. One night, there was a power failure. The disk where the database was stored was damaged. The database is not accessible anymore, and you want to recover the database. The database has the following tablespaces:

- SYSCATSPACE (system catalogs)
- USERSPACE1 (user data table space)
- USERSPACE2 (user data table space)
- USERSPACE3 (user data table space)

The following are available to you:

- All the log files. Because the logs are stored on a separate disk from the database, they were unharmed.
- You do *not* have any database level backups, but you have the following tablespace backups:
 - TEST.3.DB2.NODE0000.CATN0000.20060515135047.001 - Backup of SYSCATSPACE and USERSPACE1
 - TEST.3.DB2.NODE0000.CATN0000.20060516135136.001 - Backup of USERSPACE2 and USERSPACE3
 - TEST.3.DB2.NODE0000.CATN0000.20060517135208.001 - Backup of USERSPACE3

If we were to use the restore and rollforward methods (discussed in the previous sections) to recover the database to the most recent point in time, we would need to restore a database backup and then rollforward the database to end of logs. Unfortunately, in this case, this is not possible because we do not have a database backup. We only have tablespace backups. If we run a typical `RESTORE` command on any of the tablespace backups, we would get the following error:

```
db2 restore db test taken at 20060517135208
SQL2560N The target database is not identical to the source database for a restore
from a table space level backup.
```

With the database rebuild function, we now can rebuild the TEST database with only tablespace backups and logs. To rebuild a database, specify the `REBUILD` option in the `RESTORE DATABASE` command.

The following steps rebuild the TEST database to the most recent point in time.

1. Issue a `RESTORE DATABASE` command with the `REBUILD` option:

```
db2 restore db test rebuild with all tablespaces in database taken at 20060517135208
```

The first step in a rebuild process is to identify the rebuild target image. The rebuild target image should be the most recent backup image that you want to use in your rebuild operation. It is known as the target image because it defines the structure of the database to be rebuilt, including the table spaces that can be restored, the database configuration, and the log sequence. It can be any type of backup (full, table space, incremental, online or offline). In this example, the most recent backup image is TEST.3.DB2.NODE0000.CATN0000.20060517135208.001; therefore we use it as the target image of our rebuild operation.

After this command is executed successfully, the structure of the TEST database is restored. We can get information such as the database configuration, and its history. If we issue a `LIST HISTORY` command (for example, `db2 list history all for test`), we will get the following output:

```
Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log Backup ID
-----
R D 20060519121107001 F 20060517135208
-----
Contains 1 tablespace(s):
00001 USERSPACE3
-----
Comment: RESTORE TEST WITH RF
Start Time: 20060519121107
End Time: 20060519121108
Status: A
-----
EID: 7 Location:

Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log Backup ID
-----
R P 20060519121108001 F 20060515135047
-----
Contains 2 tablespace(s):
00001 USERSPACE1
00002 SYSCATSPACE
-----
Comment: RESTORE TEST WITH RF
Start Time: 20060519121108
End Time: 20060519121113
Status: A
-----
EID: 8 Location:

Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log Backup ID
-----
R P 20060519121113001 F 20060516135136
-----
Contains 1 tablespace(s):
00001 USERSPACE2
-----
Comment: RESTORE TEST WITH RF
Start Time: 20060519121113
End Time: 20060519121114
Status: A
-----
EID: 9 Location:

Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log Backup ID
-----
R D 200605191211107 R S0000001.LOG S0000003.LOG 20060518135208
-----
Contains 4 tablespace(s):
00001 USERSPACE3
00002 USERSPACE2
00003 USERSPACE1
00004 SYSCATSPACE
-----
Comment: REBUILD TEST WITH RF
Start Time: 20060519121107
End Time: 20060519121115
Status: A
-----
EID: 10 Location:
```


There are four entries in the `LIST HISTORY` command output shown above. They are all related to the rebuild operation.

The first entry, EID: 7, indicates a restore operation on the backup image 20060517135208, and the tablespace to be restored is `USERSPACE3`. (Recall that this backup image only contains `USERSPACE3`.) However, we have asked to restore all tablespaces using the `ALL TABLESPACES` option, so the rest of the tablespace in the database will also be restored. This is reflected in the rest of the `LIST HISTORY` output.

Using the information in the backup history file, the restore utility finds the backup images of all the tablespaces to be restored and restores them. After the restore, the tablespaces are placed in roll-forward pending state. You can see the comment line in the `LIST HISTORY` output that each tablespace is flagged with 'WITH RF', which indicates a rollforward is required following the restore.

For the restore to work, all backup images must exist in the locations as stored in the history file. Otherwise, an error is returned stating the restore utility cannot find a required image.

2. Issue a `ROLLFORWARD DATABASE` command with the `TO END OF LOGS` option:

```
db2 rollforward db test to end of logs
```

After all tablespaces have been restored, they are put in rollforward pending state. We need to rollforward the database to bring the database back to a normal state.

To rollforward a database during a rebuild operation, all log files for the time frame between the earliest and most recent backup images must be available for the rollforward utility to use. If you want to rollforward to a point in time more recent than the last backup, all the log files created after the backup must also be available.

In our example, all logs are still in good shape and they are still in the log path specified by the `LOGPATH` database configuration parameter. The rollforward utility will find them there. This is why we did not need to specify the location of the logs files in the `ROLLFORWARD` command. If the logs files had been stored somewhere else, then we must specify the location of the logs files using the `OVERFLOW LOG PATH` option in the `ROLLFORWARD` command.

3. Issue a `ROLLFORWARD DATABASE` command with the `STOP` option:

```
db2 rollforward db test stop
```

At this point the `TEST` database is connectable and all table spaces are in `NORMAL` state.

Rebuilding a recoverable database using only a subset of tablespace backups

As demonstrated by the previous example, the database rebuild functions let us rebuild an entire database using only tablespace backups and logs. What makes this utility so robust is that we do not need to have all tablespace backups to rebuild a database. We can rebuild a database with only a subset of tablespace backups.

Let's reuse our last example, and say that the data in `USERSPACE1` and `USERSPACE2` are really important to our users. We must restore these two tablespaces as soon as possible following the power failure. `Userspace3` is not as important and it is huge. If we restore all the tablespaces, it's going to take a long time. It would be nice if we can rebuild a connectable database with only

USERSPACE1 and USERSPACE2 in it, so users can use the database right away. When time permits, we can then restore USERSPACE3. The following steps show how this can be done using the database rebuild utility.

1. Issue a `RESTORE DATABASE` command with the `REBUILD` option, specifying only a subset of the tablespaces that you want restored:

```
db2 restore db test rebuild with tablespace (SYSCATSPACE,USERSPACE1,USERSPACE2)
taken at 20060516135136
```

Although we only wanted to restore USERSPACE1 and USERSPACE2, we must restore SYSCATSPACE as well, because this tablespace holds all the system information. Without it, DB2 would not know anything about the structure of this database.

The target image we specified in the above command is the image that contains USERSPACE2 and USERSPACE3. This is the most recent backup that contains the tablespaces we want to restore. Although, image 20060517135208 is the latest backup of the three, we cannot use it because it does not contain USERSPACE1, USERSPACE2, or SYSCATSPACE.

The following command has the same effect:

```
db2 restore db test rebuild with all tablespaces in database except tablespace
(USERSPACE3) taken at 20060516135136
```

2. Issue a `ROLLFORWARD DATABASE` command with the `TO END OF LOGS` option:

```
db2 rollforward db test to end of logs
```

3. Issue a `ROLLFORWARD DATABASE` command with the `STOP` option:

```
db2 rollforward db test stop
```

You may choose to rollforward to a point in time instead of end of logs. The point in time you choose must be greater than the timestamp of the backup image you used in the restore.

At this point, the TEST database is connectable and all tablespace are in NORMAL state except USERSPACE3. USERSPACE3 is in RESTORE PENDING state.

You can restore USERSPACE3 at a later time, using a normal tablespace restore (without the `REBUILD` option):

1. Issue a `RESTORE DATABASE` command and specify the tablespace to be restored:

```
db2 restore db test tablespace (USERSPACE3) taken at 20060517135208
```

2. Issue a `ROLLFORWARD DATABASE` command with the `TO END OF LOGS` option and specify the tablespace to be rolled forward:

```
db2 rollforward db test to end of logs tablespace (USERSPACE3)
```

3. Issue a `ROLLFORWARD DATABASE` command with the `STOP` option:

```
db2 rollforward db test stop
```

Now all four tablespaces of the TEST database are in NORMAL state.

Bringing only a subset of table spaces online is useful in a production environment, or in a recovery situation like the above. It is also useful in a test environment, where you only need to restore a subset of tablespaces for interested parties.

Rebuilding a recoverable database using online backup images that contain log files

When rebuilding a recoverable database, you can use either database backups or tablespace backups. The backups can also be either online or offline.

If you have an online backup image that contains log files, and you wish to use these logs to rollforward the database, you can retrieve the logs from the image using the `LOGTARGET` option of the `RESTORE DATABASE` command.

Let's reuse our TEST database as an example, and assume that the backup image TEST.3.DB2.NODE0000.CATN0000.20060517135208.001 was an online backup image that included logs. To recover the entire database using the tablespace backups and the logs that are stored in the backup image:

1. Issue a `RESTORE DATABASE` command with the `LOGTARGET` option. During the restore, the logs are extracted to the location specified by `LOGTARGET`.

```
db2 restore db test rebuild with all tablespaces in database taken at 20060517135208  
logtarget /logs
```

2. Issue a `ROLLFORWARD DATABASE` command with the `TO END OF LOGS` option and specify the location of the logs:

```
db2 rollforward db test to end of logs overflow log path (/logs)
```

Note the `OVERFLOW LOG PATH` option is used to specify the log location.

3. Issue a `ROLLFORWARD DATABASE` command with the `STOP` option:

```
db2 rollforward db test stop
```

Rebuilding a recoverable database using incremental backup images

Incremental backup images can also be used to rebuild a database. When an incremental image is involved in a rebuild process, by default the restore utility tries to use automatic incremental restore for all incremental images. If you do not use the `INCREMENTAL AUTOMATIC` option of the `RESTORE DATABASE` command, but the target image is an incremental backup image, the restore utility will issue the rebuild operation using automatic incremental restore.

If the target image is not an incremental image, but another required image is an incremental image, then the restore utility will also make sure those incremental images are restored using automatic incremental restore. The restore utility will behave in the same way whether you specify the `INCREMENTAL AUTOMATIC` options or not.

If you specify the `INCREMENTAL` option without the `AUTOMATIC` option, you will need to perform the entire rebuild process manually. The restore utility will just restore the initial metadata from the target image, as it would in a regular manual incremental restore. You will then need to complete the restore of the target image using the required incremental restore chain (see [Incremental restore](#)). Then you will need to restore the remaining images to rebuild the database. This process can be cumbersome.

It is recommended that you use automatic incremental restore to rebuild your database. Only in the event of a restore failure should you try to rebuild a database using manual methods.

Rebuilding a recoverable database using the redirect option

Since the rebuild functions are part of the restore utility, you can rebuild a database using the redirect method, as in redirected restore. The following rebuilds the entire TEST database to the most recent point in time using the `REDIRECT` option:

1. Issue a `RESTORE DATABASE` command with the `REBUILD` and `REDIRECT` option:

```
db2 restore db test rebuild with all tablespaces in database taken at 20060517135208
redirect
```

2. Issue a `SET TABLESPACE CONTAINERS` command for each table space whose containers you want to redefine. For example:

```
db2 set tablespace containers for 3 using (file '/newuserspace2' 10000)
db2 set tablespace containers for 4 using (file '/newuserspace3' 15000)
```

3. Issue a `RESTORE DATABASE` command with the `CONTINUE` option:

```
db2 restore db test continue
```

4. Issue a `ROLLFORWARD DATABASE` command with the `TO END OF LOGS` option. (This assumes all logs are accessible in the logpath directory; otherwise, use the `OVERFLOW LOG PATH` option to specify the alternate log path.)

```
db2 rollforward db test to end of logs
```

5. Issue a `ROLLFORWARD DATABASE` command with the `STOP` option:

```
db2 rollforward db test stop
```

At this point, the database is connectable and all table spaces are in NORMAL state.

Rebuilding a non-recoverable database

All rebuild methods we've discussed so far work for non-recoverable databases as well. The only differences are:

- If a database is non-recoverable, you can only use a database backup as the target image in the rebuild operation, since tablespace backups are not available to non-recoverable databases.
- When the restore completes you can connect to the database right away -- no rollforward operation is required. However, any table spaces not restored are put in drop pending state, and they can no longer be recovered.

Let's look at an example. Suppose we have a non-recoverable database MYDB. MYDB has three tablespaces: SYSCATSPACE, USERSP1 and USERSP2. A full database backup was taken at 20060521130000.

To rebuild the database using only SYSCATSPACE and USERSP1:

```
db2 restore db mydb rebuild with tablespace (SYSCATSPACE, USERSP1) taken at
20060521130000
```

Following the restore, the database is connectable. If you issue the `LIST TABLESPACES` command you will see that the `SYSCATSPACE` and `USERSP1` are in `NORMAL` state, while `USERSP2` is in `DROP PENDING` state. You can now work with the two table spaces that are in `NORMAL` state.

If you want to take a database backup, you must first drop `USERSP2` using the `DROP TABLESPACE` command or the backup will fail.

Database rebuild restrictions

The ability to rebuild a database makes the restore utility more robust. However, there are a few restrictions:

- One of the table spaces you rebuild must be `SYSCATSPACE`.
- You cannot perform a rebuild operation using the Control Center GUI tools. You must either issue commands using the command line processor (CLP), or use the corresponding application programming interfaces (APIs).
- The `REBUILD` option cannot be used against a pre-Version 9.1 target image unless the image is that of an offline database backup. If the target image is an offline database backup, then only the table spaces in this image can be used for the rebuild. The database will need to be migrated after the rebuild operation successfully completes. Attempts to rebuild using any other type of pre-Version 9.1 target image will result in an error.
- The `REBUILD` option cannot be issued against a target image from a different operating system than the one being restored on unless the target image is a full database backup. If the target image is a full database backup, then only the table spaces in this image can be used for the rebuild.

Index re-creation

Rebuilding indexes

If a database crashes for some hardware or operating system reason, it is possible that some indexes may be marked as invalid during the database restart phase. The configuration parameter `INDEXREC` indicates when DB2 will attempt to rebuild invalid indexes.

`INDEXREC` is defined in both the database manager and database configuration files. There are three possible settings for this parameter:

- **SYSTEM:** This value can only be specified in the database configuration file. When `INDEXREC` is set to this value, DB2 will look for the `INDEXREC` setting specified in the database manager configuration file and use this value.
- **ACCESS:** Invalid indexes are rebuilt when the index is first accessed.
- **RESTART:** Invalid indexes are rebuilt during a database restart.

Summary

So that's the story on high availability backup and recovery. In this tutorial you learned:

- Recovery concepts and the importance of having a recovery plan.
- What a transaction (unit of work) is and how working with transactions ensures data integrity.

- The three types of recovery: crash, version, and rollforward.

You are now familiar with DB2 transaction logs concepts:

- The log buffer
- Primary and secondary logs
- Active, online archive, and offline archive logs
- Database configuration parameters: `LOGPRIMARY`, `LOGSECOND`, and `LOGFILSIZ`, among others
- The two types of logging: circular logging and archival logging (for which `LOGARCHMETH1` database configuration parameters must be enabled)
- Infinite logging
- Recoverable and non-recoverable databases

You learned about database and tablespace backup concepts:

- How to use the backup utility
- Incremental and delta backups
- How to invoke the `BACKUP` utility from the Control Center
- The naming convention used for backup files

You learned about database and tablespace recovery concepts:

- How to use the `RESTORE` utility
- Minimum point in time
- How to invoke the `RESTORE` utility from the Control Center
- What the `QUIESCE` utility can do for you
- What the history file contains
- How to perform a redirected restore

You learned about database and tablespace rollforward concepts:

- How to use the `ROLLFORWARD` utility
- How to restore to a point in time using the `ROLLFORWARD` utility
- How to invoke the `ROLLFORWARD` utility from the Control Center
- How to use the `RECOVER` utility
- How to rebuild a database using the `RESTORE` utility
- Index re-creation and the `INDEXREC` configuration parameter

[Part 7](#) continues with high availability, and discusses split mirroring and high-availability disaster recovery.

To keep an eye on this series, bookmark the page, [DB2 9 DBA exam 731 prep tutorials](#).

© Copyright IBM Corporation 2006
(www.ibm.com/legal/copytrade.shtml)

[Trademarks](#)

(www.ibm.com/developerworks/ibm/trademarks/)

