

Backup and Recovery

Presented by DB2 Developer Domain

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

Table of Contents

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

1. Introduction	2
2. Database recovery concepts	4
3. DB2 logs	6
4. Database and table space backup	14
5. Database and table space recovery	21
6. Database and table space rollforward	27
7. Index re-creation	34
8. Conclusion	35

Section 1. Introduction

What this tutorial is about

This tutorial discusses backup and recovery topics. It explains the different methods of recovery and logging, and details how to use the `BACKUP`, `RESTORE`, and `ROLLFORWARD` commands. In this tutorial, you will learn:

- The recovery methods available with DB2®
- The concept of transaction logs and what different types of logs are available
- The types of logging methods that can be used
- How to perform `BACKUP` operations
- How to perform `RESTORE` operations
- How to perform `ROLLFORWARD` operations
- Index re-creation issues

This is the last in a series of six tutorials that you can use to help prepare for the DB2 V8.1 for Linux, UNIX®, and Windows™ Database Administration Certification (Exam 701). The material in this tutorial primarily covers the objectives in Section 6 of the exam, entitled "Backup and Recovery." You can view these objectives at: <http://www.ibm.com/certify/tests/obj701.shtml>.

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

Who should take this tutorial

In order to understand the material presented in this tutorial you should be familiar with the following:

- The DB2 environment (database manager configuration files, database configuration files, DB2 registry variables, etc.)
- Use of the Command Line processor and DB2 GUI tools to invoke DB2 commands
- The different DB2 objects (buffer pools, table spaces, tables, indexes, etc.)
- Basic SQL operations that can be performed against a database (`UPDATE`, `INSERT`, `DELETE`, and `SELECT SQL` statements)

You can obtain the above skills by completing the [tutorials for the DB2 Fundamentals](#)

Exam (Exam 700).

This tutorial is one of the tools to help you prepare for Exam 701. You should also review the resources at the end of this tutorial for more information about backup and recovery (see [Resources](#) on page 36).

About the author

Raul F. Chong is a database consultant from the IBM Toronto Laboratory who works primarily with IBM business partners. Raul has worked for five years at IBM, three of them in DB2 technical support, and two as a consultant specializing in database performance tuning, database application development, and migrations from other RDBMSs to DB2.

Raul has written several articles about DB2 for the different supported platforms, including DB2 UDB for z/OS® and OS/390®, DB2 UDB for iSeries, and, of course, DB2 for Linux, UNIX, and Windows. These articles have been published at [the DB2 Developer Domain Web site](#). You can use the search field in that Web site to review them; just type `Raul` as the keyword for your search.

You can reach Raul at rfchong@ca.ibm.com.

Acknowledgements

I would like to thank Dwaine R. Snow and Clara Liu for taking the time to review the material presented in this tutorial. Dwaine has ample experience with the DB2 product and has written several books about it. Clara also has extensive experience with DB2, and has taught the DB2 Certification preparation course to many IBM business partners and customers. Both have provided very valuable comments!

Section 2. 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 may inadvertently be corrupting 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?

In order 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 may be damaged by fire, flooding, or other similar disasters.
-

Recovery strategies

In order 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 known as 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. A typical example used to explain this concept is a customer trying to transfer \$100 from a savings account to a checking account. The UOW in this case would look like this:

```
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! Since 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 that held prior to the beginning of the transaction.

Note that 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

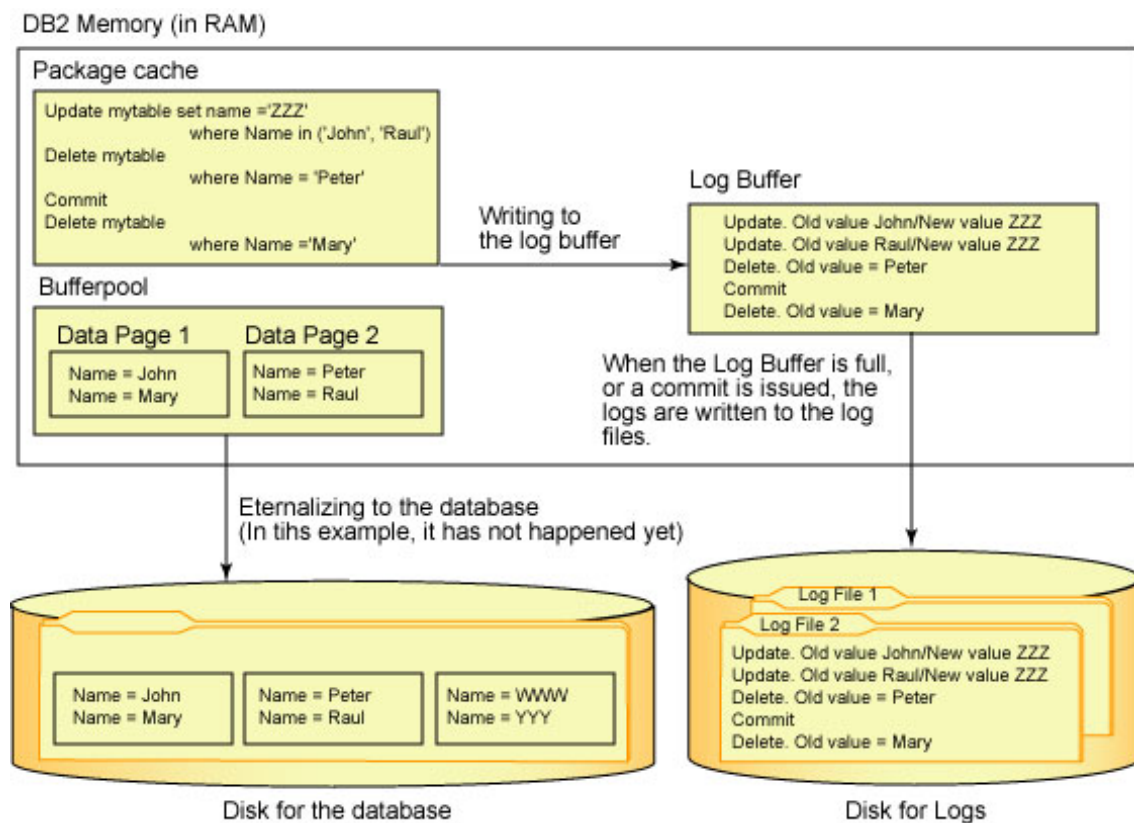
Let's familiarize ourselves with the types of recovery concepts. DB2 allows for the following types of recovery:

- **Crash recovery.** This type of 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. (Note that the order in which statements are rolled back is the reverse of the order in which they were originally executed).
- **Version recovery.** This type of 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.** This type of 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 table space to a particular point in time. Rollforward recovery requires *archival logging* to be enabled. Archival logging is discussed in a later section of this tutorial (see [Types of logging](#) on page 9).

Section 3. DB2 logs

Understanding DB2 logs

DB2 transaction logs are crucial 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. In order 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 illustrates 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:

- **LOGPRIMARY**: This parameter indicates the number of primary log files to be allocated.
- **LOGSECOND**: This parameter indicates the maximum number of secondary log files that can be allocated.
- **LOGFILSIZ**: This parameter 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) = 250
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 250 4 KB pages, are allocated. If you look in the C:\mylogs directory, you will see the three files:

```
Directory of C:\MYLOGS\
2003-03-10  06:06p           1,032,192  S0000000.LOG
2003-03-10  06:06p           1,032,192  S0000001.LOG
2003-03-10  06:06p           1,032,192  S0000002.LOG
                3 File(s)          3,096,576 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;
```

Earlier, 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.

Infinite logging

Can't you just make LOGSECOND larger to avoid running out of log space? The maximum number of secondary logs is 254. You don't want to specify a large number, however, since there is a performance price associated with the allocation of files. Typically, you want to specify enough LOGSECOND log files to handle a spike in your load (such as a heavier transaction load at the end of a month). Secondary log files will not be removed until the next time the database is activated (or on the first connection after all connections have disconnected).

To allow infinite *active* logging:

1. Set the USEREXIT database configuration parameter to ON.
2. Set LOGSECOND to a value of -1

Note the use of the term *active* above. The concepts of *active* and *archive* logs are discussed in the next panel.

Types of logs

In this panel, we'll briefly define the different types of logs. In the next panel, you'll see how they are used when describing circular and archival 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.** This type of log contains information for committed *and* externalized transactions. Such logs are kept in the same directory as the active logs.

- **Offline archive logs.** These are the archive logs that have been moved from the active log directory to another directory or media. This move can be done either manually or as an automated process using userexits.

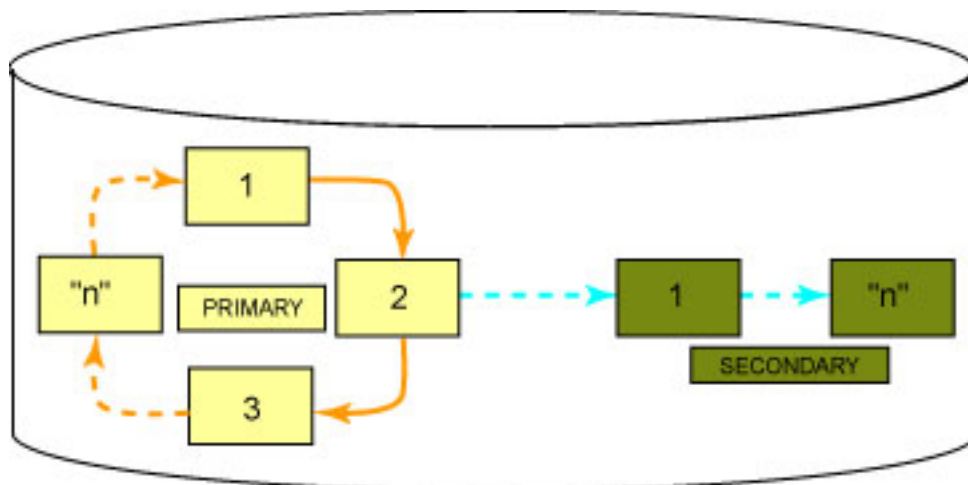
Types of logging

There are two types of logging:

Circular logging. Circular logging is the default logging mode for DB2. As you can tell from the name, 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, etc.

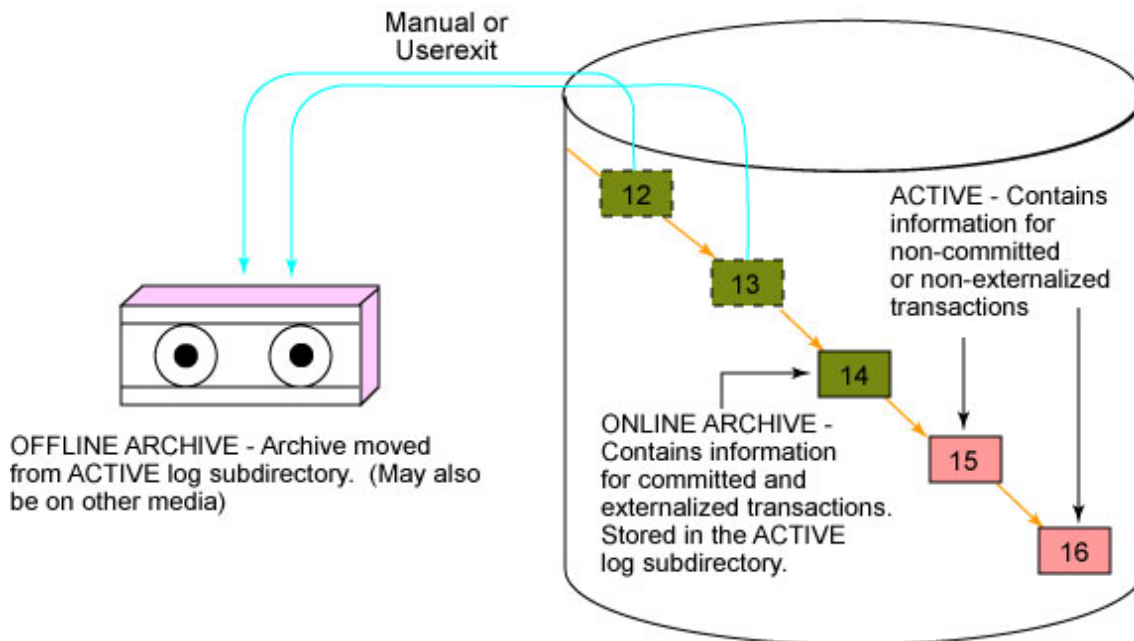
A log can be reused in circular logging as long as it only contains information about transactions that have already been committed *and* externalized to the database disk. In other words, if the log is still an active log, it cannot be reused.

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 previous section (see [Primary and secondary log files](#) on page 7). The figure below illustrates how this works:



Archival logging. Again, as you can tell from the name, 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, etc.

As you can see from 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. In other words, it will not overwrite the logs that have become archive logs. The figure below illustrates how this works:



This figure is self-explanatory, and summarizes several of the concepts we have covered so far.

Note that archival logging needs to be enabled before it can be used. To enable it, you have to turn on either or both of the following parameters:

```
LOGRETAIN (db2 update db cfg for database_name using LOGRETAIN ON)
USEREXIT (db2 update db cfg for database_name using USEREXIT ON)
```

Other recovery topics

Logging types vs. 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 vs. nonrecoverable databases

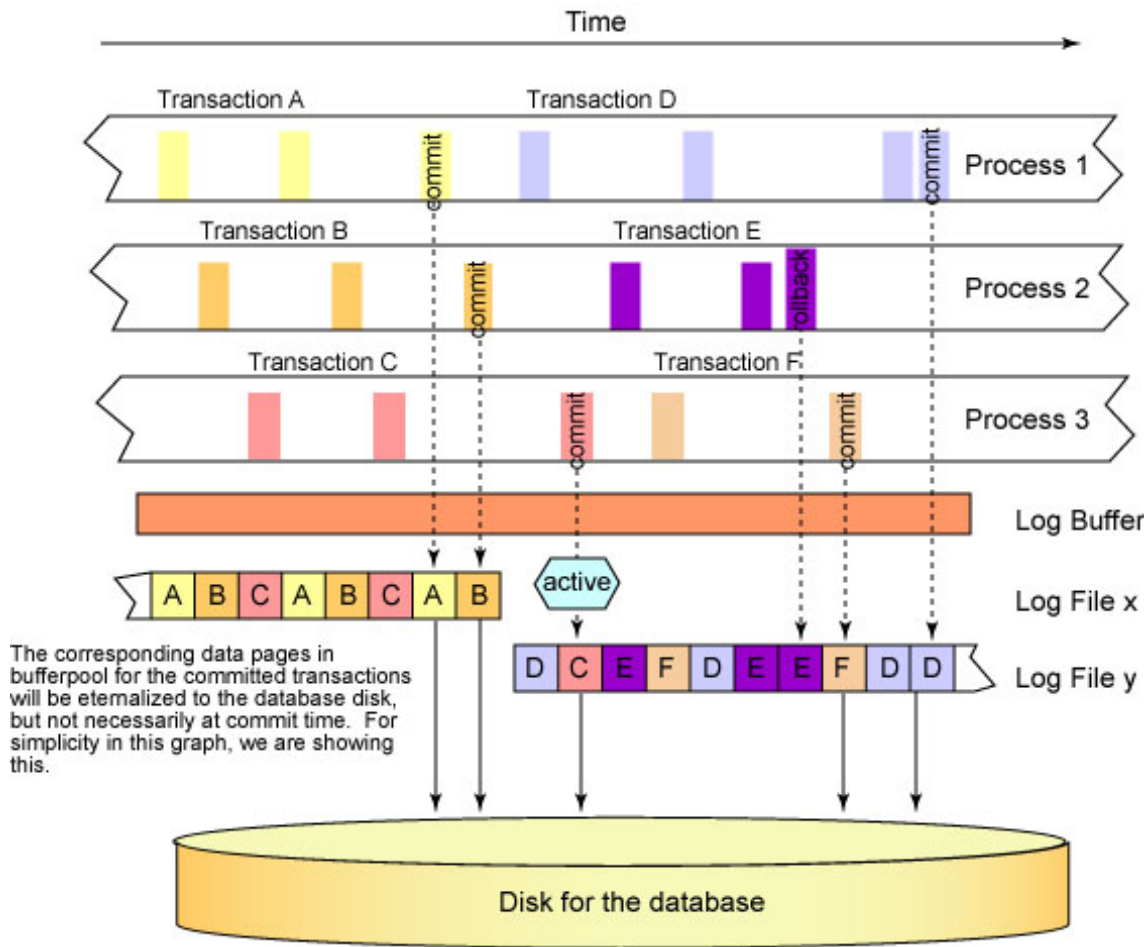
Recoverable databases are databases that can be recovered using crash, version, or rollforward recovery; thus, archival logging needs to be enabled for these databases. *Nonrecoverable* databases are those that do not support rollforward recovery; thus, only circular logging is used.

Userexits

We mentioned *userexits* several times in the previous sections. Userexits are programs that allow online archive logs to be moved to a directory different than the active log directory, or to another media. Userexits will also retrieve offline archive logs to the active log directory when they are needed during a `ROLLFORWARD` operation for a full database restore. To enable userexits, set the `USEREXIT` database configuration parameter to `ON`. Once enabled, DB2 will automatically invoke the userexit program when needed. This program needs to be named `db2uext2` and it should be stored in the `sql1lib\bin` directory in Windows, and `sql1lib/bin` in UNIX.

Review

We have covered several concepts about database logs and logging so far. The following figure summarizes some of these 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.

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. (MINCOMMIT is a database configuration file parameter.) 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.

Note the hexagon with the *active* label. This 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 is still considered active even after it has filled up? It's because it contains transactions that have not yet been committed and externalized. As you can see, 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.

Section 4. Database and table space backup

Online vs. offline access

We'll use the terms *online* and *offline* quite often in the following panels.

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.

Database backup

A *database backup* is a complete copy of your database. Besides the data, a backup copy contains information about the table spaces, 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.

Here's 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]] |
      TO dir/dev [ {,dir/dev} ... ] | LOAD lib-name [OPEN num-sess SESSIONS]]
      [WITH num-buff BUFFERS] [BUFFER buffer-size] [PARALLELISM n]
      [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 that command 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.

Note that 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`, as shown below:

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

We mentioned before that an online backup allows other users to access the database while it is being backed up. It is likely that some of the changes made by these users will not be stored in the backup copy when it is done. Thus, an online backup and a complete set of archived logs are required for recovery. Moreover, as soon as an online backup finishes, DB2 forces the current active log to close. As a result, this log will be archived.

Note that an online backup requires that archive logging be enabled for the database.

Table space backup

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

To perform a table space 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 table space backup as opposed to a full database backup. You can also see from the example that you can include as many table spaces in the backup as you'd like. Temporary table spaces cannot be backed up using a table space level backup.

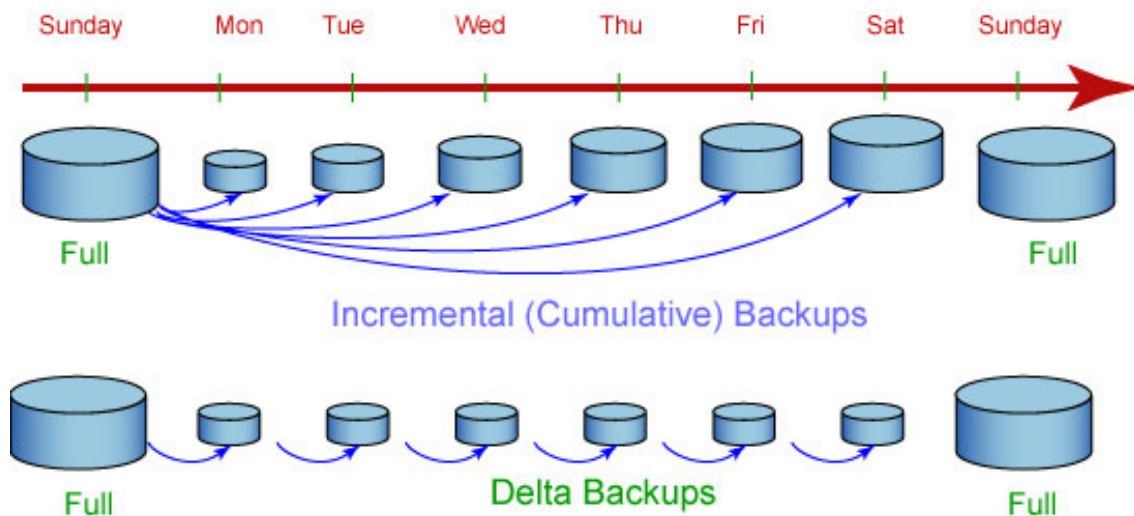
You would normally want to back up related table spaces together. For example, imagine you're using DMS table spaces, where one table space is used for the table data, another for the indexes, and another for the LOBs. You should back up all of these table spaces at the same time so that you have consistent information. This is also true for table spaces 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 illustrates the differences between these types:

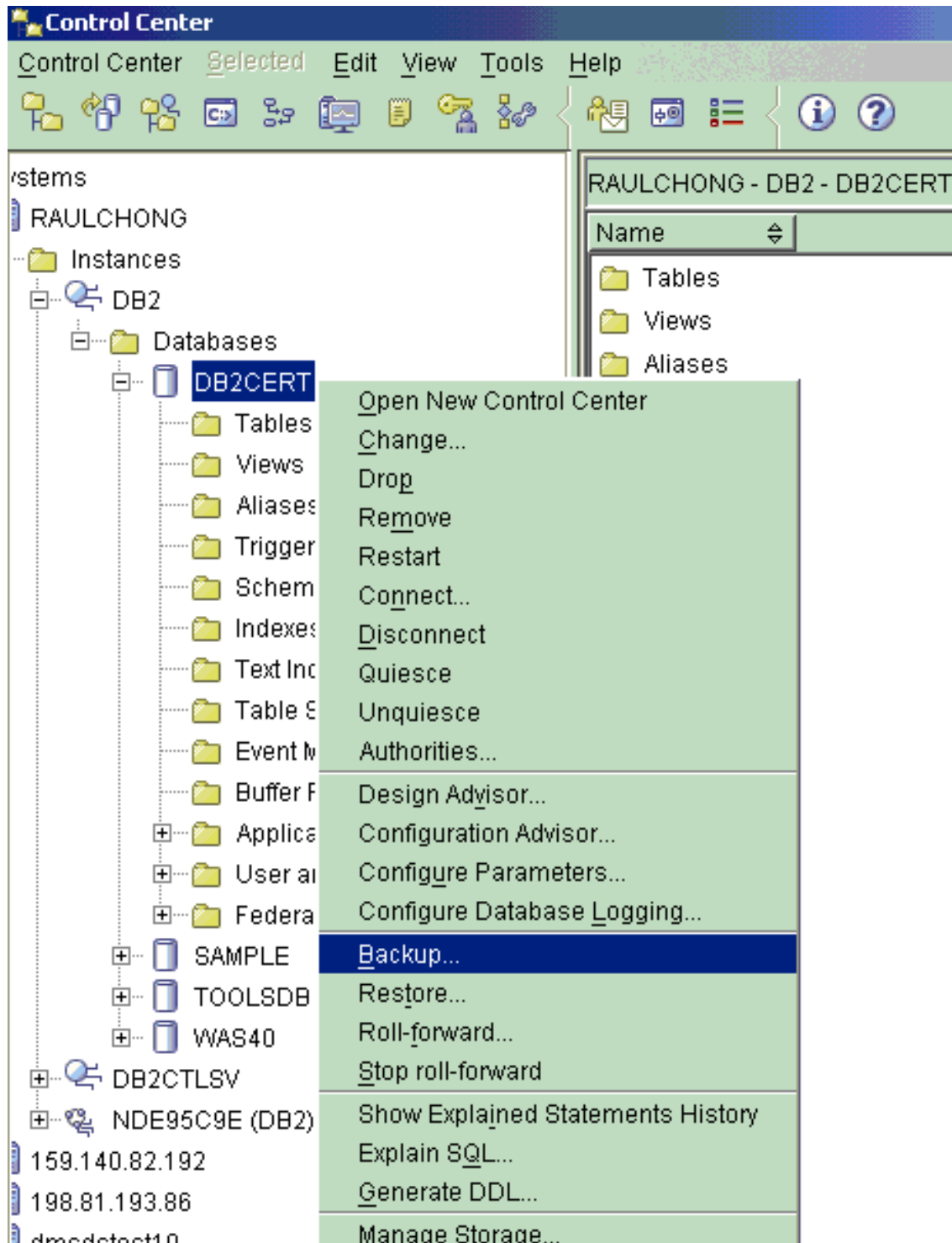


In the upper part of figure, if there were a crash after the incremental backup on Friday, you could restore the first Sunday full backup, followed by the incremental backup taken on Friday.

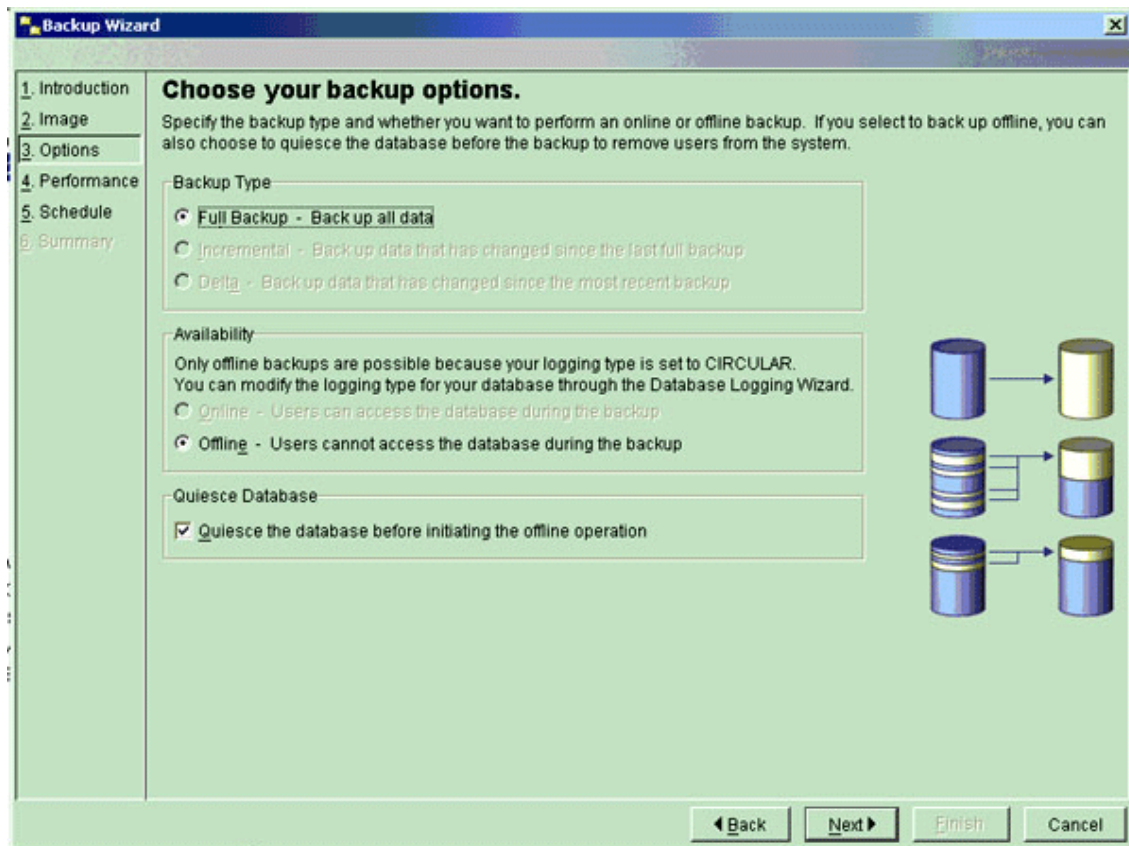
In the lower part of the figure, if there were a crash after the delta backup on Friday, you could restore the first Sunday full backup followed by each of the delta backups taken from Monday until Friday inclusive.

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 table space backup, right-click the database you want to back up and select **Backup**.



The next figure shows the options you need to fill out to execute the BACKUP utility. We have skipped some intermediate screens, in which you can choose whether you want to back up table spaces or a database. We encourage you to try this on your own.



The backup files

The naming convention for DB2 backup files on disk contain the following items:

- Database alias
- Digit indicating the type of backup (0 for a full database, 3 for a table space 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 exact naming convention varies slightly by platform, as shown in the following figure:

Example of Backup File Names

Windows

Alias Instance Year Day Minute Sequence
DBALIAS.0\DB2INST\NODE0000\CATN0000\20030314\131259.001
Type Node Catalog Node Month Hour Second

UNIX/Linux

Alias Instance Year Day Minute Sequence
DBALIAS.0.DB2INST.NODE0000.CATN0000.20030314131259.001
Type Node Catalog Node Month Hour Second

Section 5. Database and table space recovery

Database recovery

In this section, we'll discuss the `RESTORE` utility, which uses a backup file as input and a new or existing database as output. Note that 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]] [{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]
  [WITH num-buff BUFFERS] [BUFFER buffer-size]
  [DLREPORT file-name] [REPLACE EXISTING] [REDIRECT] [PARALLELISM n]
  [WITHOUT ROLLING FORWARD] [WITHOUT DATALINK] [WITHOUT PROMPTING]
```

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 20030314131259
(4) WITHOUT ROLLING FORWARD
(5) WITHOUT PROMPTING
```

Let's look at that command in more detail:

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

Table space recovery

You can `RESTORE` table spaces either from a full database backup or from a table space backup. Table space 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 table space `RESTORE` command:

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

Let's look at that command in more detail:

1. Indicates the name of the database image to restore.
2. Indicates that this is a table space `RESTORE`, and specifies the name of the table space(s) to restore.
3. Indicates that this is an online restore. Note that for user table spaces, 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.

Table space recovery considerations

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

Consider an example:

1. Imagine that at time `t1` you took a full database backup, which included the table space `mytbls1`.
2. At time `t2`, you created the table `myTable` in the table space `mytbls1`. This would set the minimum PIT for recovery of the table space `mytbls1` to `t2`.
3. At time `t3`, you decided to restore only table space `mytbls1` from the full database backup taken at `t1`.

4. After the restore is complete, table space mytbls1 will be placed in rollforward pending state. If you were allowed to roll forward to a point prior to the minimum PIT, table space mytbls1 will not have the table myTable; however, the system catalog would say that the table does exist in mytbls1. Therefore, in order to avoid inconsistencies like this, DB2 will force you to roll forward at least to the minimum PIT when you restore a table space.

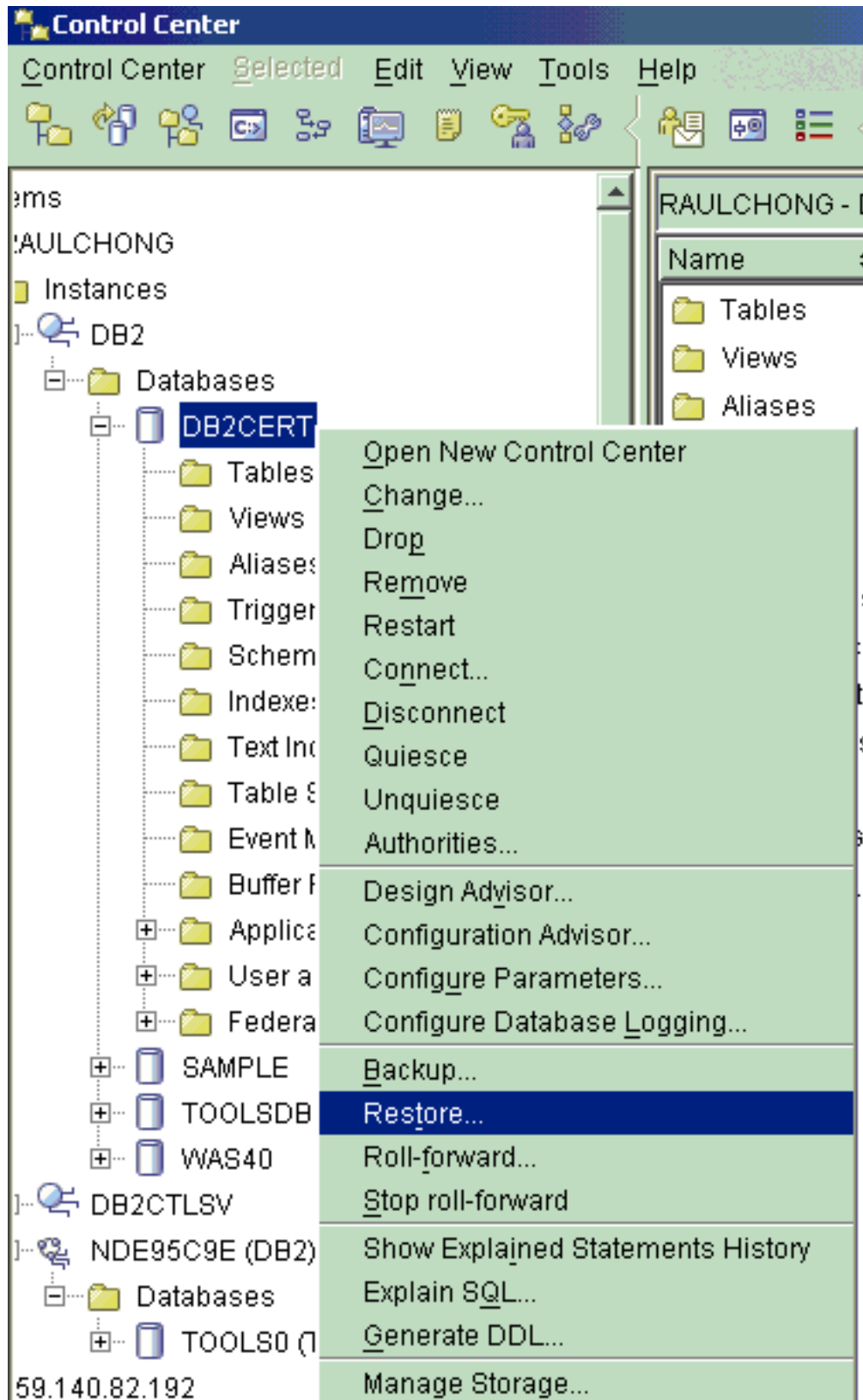
A minimum PIT is updated when DDL statements are run against the table space, or against tables in the table space. In order to determine the minimum PIT of recovery for a table space, you can use either of the following methods:

- Use the `LIST TABLESPACES SHOW DETAIL` command.
- Obtain a table space snapshot via the `GET SNAPSHOT FOR TABLESPACE ON db_name` command.

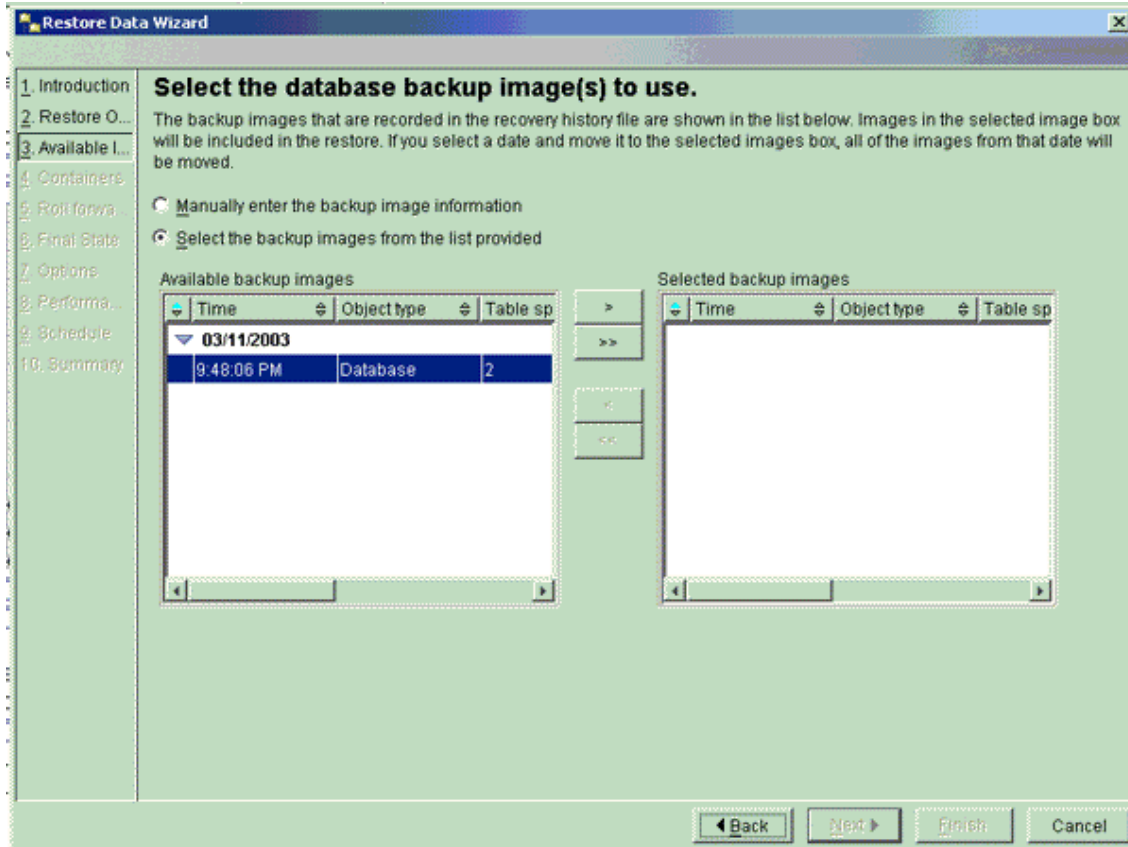
In addition, the system catalog table space (SYSCATSPACE) must be rolled forward to the end of logs and in offline mode. We'll discuss more about the `ROLLFORWARD` command in the next section (see [Database and table space rollforward](#) on page 27).

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 table space restore, right-click the database you want to restore and select **Restore**.



The next figure shows some options you need to fill out to execute the `RESTORE` utility. We have skipped some intermediate screens in which you can choose whether you want to restore table spaces or a database. We encourage you to try this on your own.



Redirected restore

We mentioned earlier that a backup file includes information about table spaces 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:

1. Obtains the information about the containers and table spaces 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:

```
SQL1277N Restore has detected that one or more table space containers are
inaccessible, or has set their state to 'storage must be defined'.
DB20000I The RESTORE DATABASE command completed successfully.
```

2. Reviews the table space information from the (partially) restored database `newdb`:

```
LIST TABLESPACES SHOW DETAIL
```

3. Sets the new containers for each table space. A table space 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")
```

In the example above, `n` represents an ID of one of the table spaces in the backup. Note also that with redirected restores, you cannot change the type of the table space; that is, if the table spaces is SMS, it cannot be changed to DMS.

4. Starts 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 table spaces. If you reviewed the second tutorial in this series, you should know that in most cases SMS table spaces cannot be altered to add a container. A redirected restore provides a workaround to this limitation.

Section 6. Database and table space rollforward

Database rollforward

In the previous section, we discussed the `ROLLFORWARD` command. In this section, we'll cover it in more detail. 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 table space to any point in time after the minimum PIT, there is no guarantee that the end time you choose to roll forward will have all data in a consistent state.

We will not cover the `QUIESCE` command in this tutorial. However, 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, is 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 `OVERFLOWLOGPATH` option and use the logs in that location if the log file is not found in the current path.
4. Call the userexit to retrieve the log file from the archive path if the log file is not found in the current path and the `OVERFLOWLOGPATH` option is not used.
5. Call the userexit to retrieve log files only if you are rolling forward a full database restore and userexit has been enabled.
6. Reapply the transactions, once the log is in the current log path or the `OVERFLOWLOGPATH`.

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

The following is the syntax of the `ROLLFORWARD` command:

```
ROLLFORWARD DATABASE database-alias [USER username [USING password]]
[TO {isotime [ON ALL DBPARTITIONNUMS] [USING LOCAL TIME] | END OF LOGS
[On-DbPartitionNum-Clause}}] [AND {COMPLETE | STOP}}] |
{COMPLETE | STOP | CANCEL | QUERY STATUS [USING LOCAL 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
```

Let's look at each statement in detail:

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.

Note that there is no keyword `OFFLINE` in the syntax, as this is the default mode. In fact, for the `ROLLFORWARD` command, this is the only mode allowed for databases.

Table space rollforward

Table space rollforwards can generally be either online or offline. The exception is the system catalog table space (`SYSCATSPACE`), which can only be rolled forward offline.

Here's an example table space 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 table space to be rolled forward.

Table space rollforward considerations

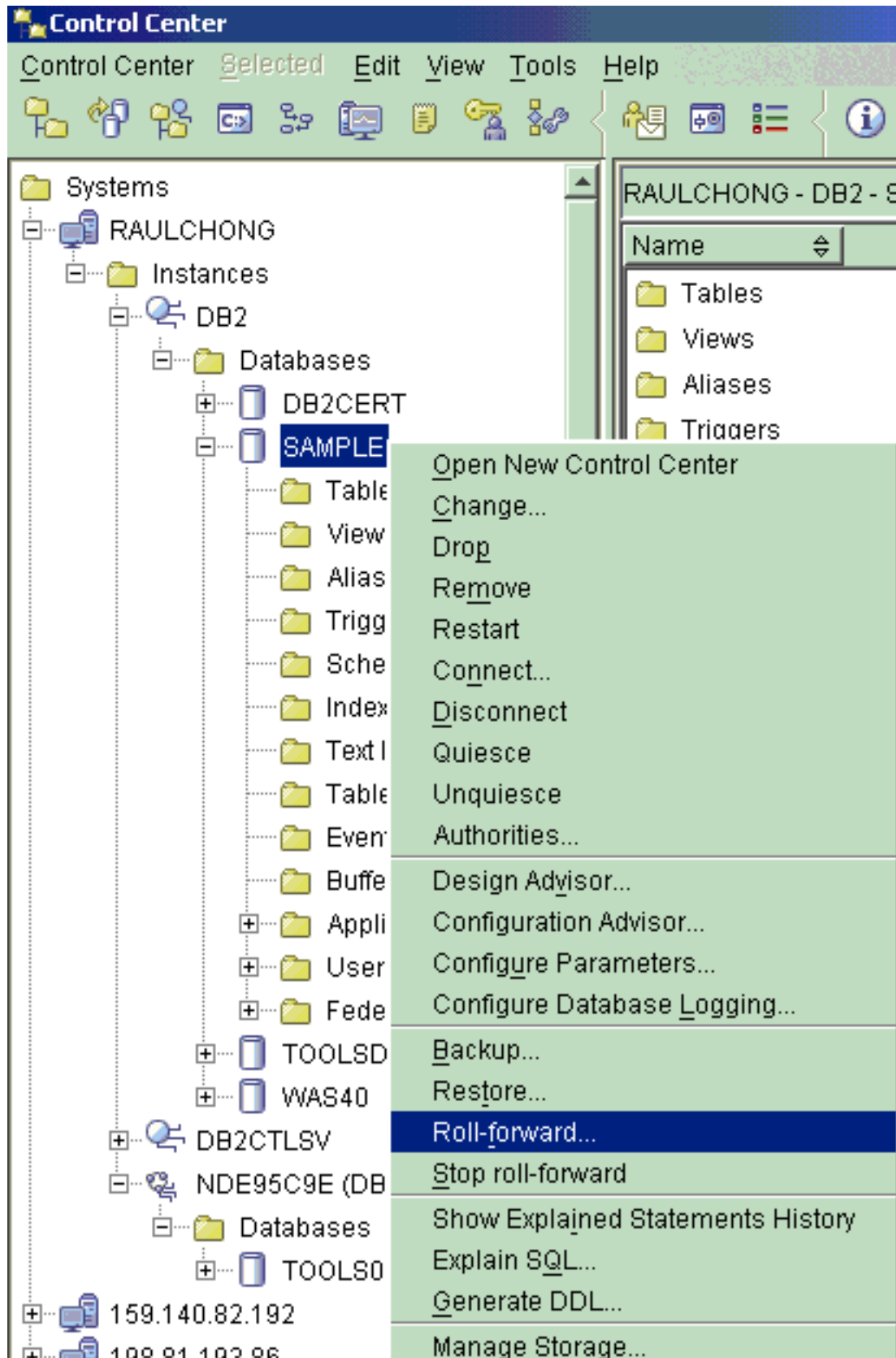
- If the registry variable `DB2_COLLECT_TS_REC_INFO` is enabled, only the log files required to recover the table space are processed. The `ROLLFORWARD` command will skip over log files that are not required, which can speed up recovery time.
- The `QUERY STATUS` option of the `ROLLFORWARD` command can be used to list: the log files that DB2 has rolled forward, the next archive log file required, and the timestamp of the last committed transaction since rollforward processing began. For example:

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

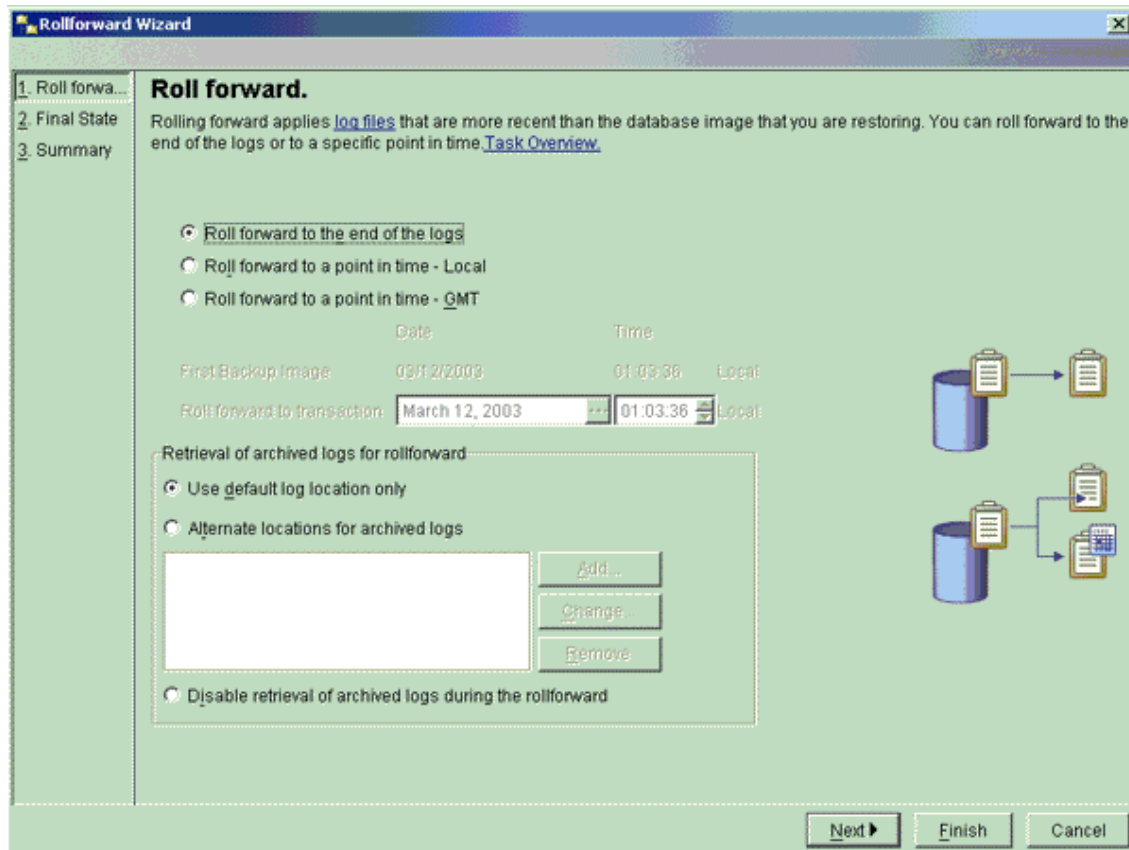
- After a table space point-in-time rollforward operation completes, the table space is put in backup-pending state. A backup of the table space or database must be taken because all updates made to it between the point in time that the table space was recovered to and the current time have been lost.

Performing rollforwards operations with the Control Center

The figure below shows you how to invoke the `ROLLFORWARD` command from the Control Center. To perform a database or table space rollforward, right-click the database you want to roll forward and select **Roll-forward**.



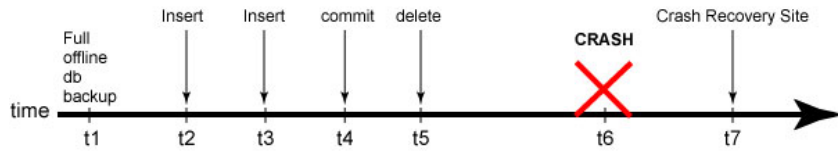
This next figure shows some options you need to fill out to execute the `ROLLFORWARD` command. We encourage you to try this on your own.



Review and examples

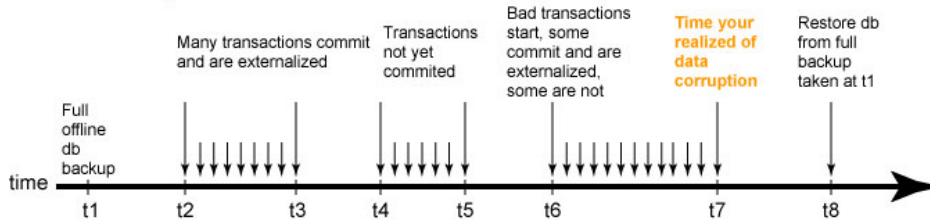
So far we have discussed the `BACKUP`, `RESTORE`, and `ROLLFORWARD` commands. The figure below reviews the different types of recovery which you should now be able to understand easily.

Crash Recovery



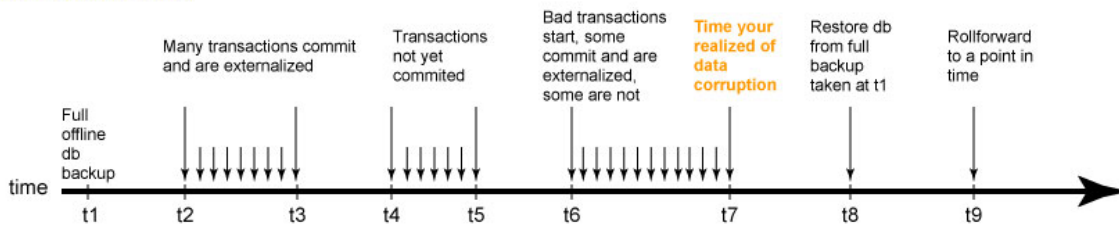
For this scenario, circular logging is in effect. At t6 there is an unscheduled power shutdown in your building. At t7, DB2 is restarted, and when you connect to the database, crash recovery is automatically started (assuming db cfg AUTORESTART is ON, otherwise you manually have to perform this with 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.

Version Recovery



For this scenario, circular logging is in effect. At t7 you realize your data in all table spaces has been corrupted by some transaction which started at t6. Thus, 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, thus you cannot even roll forward active logs). Conclusion: Many of the good transactions from t2 to t4 will be lost.

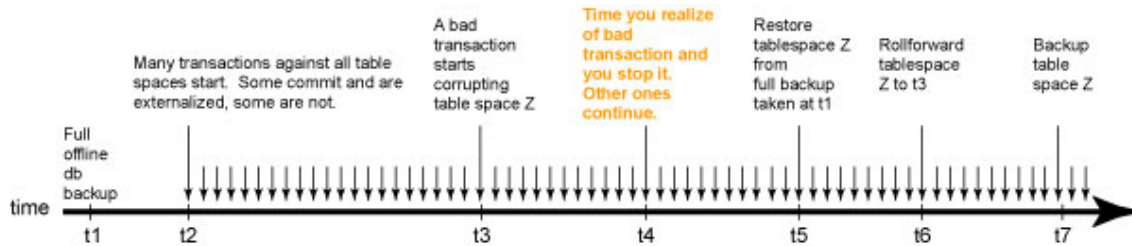
Rollforward Recovery



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); thus, after the full database restore is applied at t8, you can rollforward the logs at t9. Logs can be roll forwarded 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:

Backup, Restore and Rollforward Example



Scenario:

1. An offline database backup finished at time t1
2. Daily transactions are performed starting at t2.
3. At t4 you realize one of the transactions has corrupted table space Z, and you stop this transaction. Other transactions against other table spaces continues.
4. Only for table space 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 table space Z from the full offline backup taken at t1
 - After the restore has finished, the table space will be left in rollforward pending state.
 - At t6 you rollforward the table space 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 table space in backup pending state for consistency reasons.
 - At t7 you back up the table space. At this point, your database is consistent and all users and applications can work normally. The restored table space will have a gap from t3 to t7, which is what we intended to remove the corrupted data.

Section 7. 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:** This means invalid indexes are rebuilt when the index is first accessed.
- **RESTART:** This means invalid indexes are rebuilt during a database restart.

Section 8. Conclusion

Summary

Here is a summary of what you learned in this tutorial:

- You learned recovery concepts and the importance of having a recovery plan.
- You learned what a transaction (unit of work) is and how working with transactions ensures data integrity.
- You learned there are 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 LOGRETAIN and/or USEREXIT database configuration parameters must be enabled).
 - Recoverable vs. non-recoverable databases
 - Userexits
- You learned about database and table space 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 table space recovery concepts:
 - How to use the `RESTORE` utility
 - The concept of minimum point in time (PIT)
 - How to invoke the `RESTORE` utility from the Control Center
 - What the `QUIESCE` utility can do for you
 - What the history file contains
 - The concept of a redirected restore
- You learned about database and table space rollforward concepts:
 - How to use the `ROLLFORWARD` utility
 - How to restore to a point in time (PIT) using the `ROLLFORWARD` utility
 - How to invoke the `ROLLFORWARD` utility from the Control Center
- You learned about index re-creation and the `INDEXREC` configuration parameter.

If you feel good about the materials covered in this tutorial, you should have a good chance of doing well in section 6 of the exam 701.

Good Luck!

Resources

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

- [Server Management: DB2 V8.1 Database Administration certification prep, Part 1 of 6](#)
- [Data Placement: DB2 V8.1 Database Administration certification prep, Part 2 of 6](#)
- [Database Access: DB2 V8.1 Database Administration certification prep, Part 3 of 6](#)
- [Monitoring DB2 Activity: DB2 V8.1 Database Administration certification prep, Part 4 of 6](#)
- [DB2 Utilities: DB2 V8.1 Database Administration certification prep, Part 5 of 6](#)

For more information about backup and recovery, see the following resources:

- *Administration Guide: Planning*. International Business Machines Corporation, 2002.
- *Data Recovery and High Availability Guide and Reference*. International Business Machines Corporation, 2002.
- *Command Reference*. International Business Machines Corporation, 2002.

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

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

Feedback

Colophon

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

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