IBM DB2® 9.7
Backup and Recovery Hands-On Lab

Information Management Cloud Computing Center of Competence
IBM Canada Lab
# Contents

CONTENTS ................................................................................................................................. 1

1. INTRODUCTION .................................................................................................................. 3

2. BASIC SETUP ...................................................................................................................... 3

  2.1 Environment Setup Requirements ................................................................................. 3
  2.2 Preparation Steps ............................................................................................................ 3

3. DB2 LOGGING ..................................................................................................................... 4

  3.1 Logging Parameters ....................................................................................................... 4
    3.1.1 LOGFILSZ ............................................................................................................. 5
    3.1.2 LOGPRIMARY AND LOGSECOND ................................................................. 5
    3.1.3 LOGBUFSZ ......................................................................................................... 5
  3.2 Types of Logging .......................................................................................................... 6
    3.2.1 CIRCULAR LOGGING ....................................................................................... 6
    3.2.2 ARCHIVAL LOGGING ...................................................................................... 6

4. RECOVERY SCENARIO ........................................................................................................ 8

  4.1 Scenario 1 - Entire database is accidentally dropped or becomes corrupted .................. 8
  4.2 Scenario 2 - Database Roll forward to a Point in Time .................................................. 9
  4.3 Scenario 3 – Incremental Backup and Restore ............................................................. 11
    4.3.1 INCREMENTAL BACKUP ............................................................................... 11
    4.3.2 INCREMENTAL RESTORE .............................................................................. 13
1. Introduction

Various situations may threaten the integrity of the database including system outage, hardware failure, transaction failure, and disaster. DB2’s backup and recovery prevents you from losing data.

By the end of this lab, you will be able to:

- Perform a full backup and restore
- Restore a database to a point in time
- Perform an incremental backup and restore

2. Basic Setup

2.1 Environment Setup Requirements

To complete this lab you will need the following:

- DB2 Academic Workshop VMware® image
- VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in VMware Basics and Introduction from module 1.

2.2 Preparation Steps

1. Start the VMware image. Once loaded and prompted for login credentials, use the user “db2inst1” to provide DBADM authority:
   User: db2inst1
   Password: password

2. Type in the command “startx” to bring up the graphical environment.

3. Open a terminal window by right-clicking on the Desktop and choosing the “Open Terminal” item:

4. If the Database Manager is not yet started, issue the following command:

   db2start
For executing this lab, you will need DB2’s sample database created in its original format.

Execute the commands below to drop (if it already exists) and recreate the SAMPLE database:

```
db2 force applications all
db2 drop db sample
db2sampl
```

### 3. DB2 Logging

A transaction is a logical unit of work. Every transaction performed by DB2 is first written to the log and is then performed against the data. DB2 relies on these log files for backup and recovery.

Before we can go into the different types of DB2 logging, we first have to understand some logging parameters.

#### 3.1 Logging Parameters

To see the database configuration that is related to logging, run the following command:

```
db2 get db cfg for sample | grep LOG
```
3.1.1 LOGFILSZ

It is the size of each transactional log file measured in 4KB pages. The default size is 1000 pages or 4 MB, which implies that it can hold up to 4 MB of transactional data. You can configure it to be larger if it is going to be a high-transaction OLTP type of environment. In an OLTP environment, small-sized log files would fill up very quickly and new log files would have to be created frequently.

3.1.2 LOGPRIMARY and LOGSECOND

LOGPRIMARY is the number of primary log files. At any given time, there might be some uncommitted transactions in the database that make up the active log space. Active log space refers to the total sum of log space taken up by uncommitted transactions. By default it is 3; therefore if you have 3 log files worth of uncommitted transactions, any new transactions would start utilizing the secondary log files.

LOGSECOND is the number of secondary log files. These are allocated only when a transaction exhausts all the space configured for the primary log, to accommodate spikes in transactional activity. Once the transactions using the secondary log files commit or roll back, DB2 returns to using primary log. You can have this configured. By default LOGSECOND is 2, meaning if primary log files filled up with uncommitted transactions, 2 more log files will be allocated temporarily to handle the spike. If all the primary and secondary log files have been used, then an error will be returned:

SQL0964C The transaction log for the database is full.

Set LOGPRIMARY to 5 and LOGSECOND to 3; issue the following command from the terminal window:

db2 update database configuration for sample using LOGPRIMARY 5

A warning message may be returned:

SQL1363W One or more of the parameters submitted for immediate modification were not changed dynamically. For these configuration parameters, all applications must disconnect from this database before the changes become effective.

In order for the change of configuration to take effect, simply disconnect and reconnect to it, since this is the only connection to the database at this moment.

db2 terminate

db2 connect to sample

3.1.3 LOGBUFSZ

All log records are written in memory before getting flushed to disk. LOGBUFSZ specifies the size of this area in memory. The default of 8 4KB pages is small for most
scenarios. This parameter is critical for OLTP performance. Set the LOGBUFSZ to 256, which is a good starting number. In a real environment, take an OLTP workload and benchmark with higher LOGBUFSZ to find the optimal value.

### 3.2 Types of Logging

DB2 databases support two different logging modes: Circular and Archival.

#### 3.2.1 Circular Logging

This is DB2’s default logging technique for a newly created database. It uses primary log files in rotation up to the number of log files indicated by the LOGPRIMARY parameter. If a long-running transaction exhausts all the primary log files before completing, the transaction spills over to the secondary log files. When the work is committed, DB2 returns to the first log file and continues in a circular fashion.

Roll-forward recovery is not possible with this logging method because log files are not kept as they are constantly being overwritten. Only crash recovery and version recovery are available. If a database is using circular logging, the database can be backed up only through an offline backup.

To enable circular logging, set both LOGARCHMETH1 and LOGARCHMETH2 database configuration parameters to OFF.

#### 3.2.2 Archival Logging

In archival logging, all log files are kept; they are never overwritten. To have online backups and the ability to perform roll forward recovery, the database needs to be enabled for archival logging.

To enable archival logging, you will need to specify the value of LOGARCHMETH1 to something other than OFF. If both LOGARCHMETH1 and LOGARCHMETH2 have been specified, then archive logs are archived twice.

Infinite logging is a variation of archival logging where LOGARCHMETH2 is set to -1. With this type of logging, secondary log files are allocated until the unit of work commits.

1. We will now change the logging method to archival logging and set the archival location:

   ```
   mkdir /home/db2inst1/logarch
db2 update db cfg for sample using LOGARCHMETH1
disk:/home/db2inst1/logarch
   ```

2. Terminate the connection to the database and reconnect to the sample database:

   ```
db2 terminate
   ```
However, when you try to reconnect to the sample database, you will receive the following error:

```
SQL1116N A connection to or activation of database "SAMPLE" cannot be made because of BACKUP PENDING. SQLSTATE=57019
```

This message is received because archival logging has just been enabled for this database so it is put into backup pending state. Recall that once archival logging is enabled for the database, roll forward recoveries can be performed. However, roll forward recovery can only be performed once a backup image has been restored and database is placed in Roll Forward Pending status. Therefore, a full database backup must be made before the database can be used.

3. Create a directory to store the backup and take a full database backup by issuing the following command:

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

If no error has occurred, you will see a similar message as the following but with a different timestamp:

```
Backup successful. The timestamp for this backup image is: 20100509163937
```

When a backup image is created, the timestamp at which the backup image is created is returned in the format of `yyyyymmddhhmmss`. This is useful information because the Restore utility uses this timestamp to differentiate between multiple available backup images.

Write down the timestamp returned by your backup command, it will be referred to as T1 in following exercises.

```
T1:
```

4. Try to connect to the database again. This time it should succeed.

```
db2 connect to sample
```
4. Recovery Scenario

In this section of the lab, we will explore various scenarios in which DB2 Recovery utility can be used to recover from failure.

4.1 Scenario 1 - Entire database is accidentally dropped or becomes corrupted

If a database was accidentally dropped or is corrupted, you can recover the database by restoring a full backup.

In this example, we will restore from the offline backup image taken at the end of Exercise 3.2.2. If you had not noted down the timestamp (T1) at which the backup was taken, you can always check the Recovery history file to find the backup time stamp by issuing the following command:

```
db2 list history backup all for database sample
```

The timestamp is indicated within the circle in the screenshot below:
1. To simulate this scenario, disconnect and drop the database sample:

```
db2 force applications all
db2 drop database sample
```

If you try to connect to the sample database now, you will receive the following error

```
db2 connect to sample
```

```
SQL1013N Database alias name or database name "sample" could not found. SQLSTATE=43705
```

2. To recover from this failure, you can restore a previously created full database backup.

   Restore the database backup image that was created earlier in the previous exercise. You will need to substitute the timestamp T1 noted earlier into the command:

```
db2 restore database sample from /home/db2inst1/backups taken at <T1> without rolling forward
```

Note that there is the **without rolling forward** clause in the restore command. Since restore is from an offline backup, it is not mandatory to do a roll forward after the restore. This is useful when a roll forward is not needed and restore can finish in just one step.

After restore finishes, you should be able to connect to the sample database without having to do a roll forward explicitly.

### 4.2 Scenario 2 - Database Roll forward to a Point in Time

Roll forward is the process of applying transaction log files after a restore has been performed. For example, the last backup was taken Sunday, and the database was lost on the following Tuesday. Once the backup from Sunday is restored, transactions in log files need to be applied in order to recover transactions that were executed after the backup was taken. This is achieved by rolling forward to **END OF LOGS**.

There might be a situation where it is not desired to apply all the transactions. For example, a large set of records are deleted from the database mistakenly by the user. In such a case, in order to recover all the deleted records, rolling forward to a **POINT IN TIME** before the deletions took place would be more appropriate.

1. To simulate this scenario, we will delete some rows from tables.

Before we began, check the number of rows in the original STAFF table within the sample database:

```
db2 connect to sample
```
db2 “select count(*) from staff”

The number of rows in the STAFF table should be 35.

Now run the following commands to delete some of the data from the STAFF table:

db2 “delete from staff where dept=10”

Check the count of the STAFF table after the delete statement:

db2 “select count(*) from staff”

There should now be 31 rows in the STAFF table.

2. We will run another delete statement on the EMPLOYEE table. However, imagine that these rows were deleted accidentally.

Run the “date” command and note the timestamp before we “accidentally” issue a delete statement.

date +%F-%H.%M.%S

This timestamp will be referred to as T2, write it down as a record as this is needed for the point in time recovery:

T2:

Now check the number of rows in the original EMPLOYEE table:

db2 “select count(*) from employee”

The number of rows in the EMPLOYEE table should be 42.

Now we will accidentally delete some data from the EMPLOYEE table:

db2 “delete from employee where edlevel=12”

Check the count of the EMPLOYEE table after the delete statement:

db2 “select count(*) from employee”

There should now be 39 rows in the EMPLOYEE table.

3. The rows that you have just deleted from the EMPLOYEE table were not supposed to be removed. If we restore the database to the last full backup, then the deletion of rows to the STAFF table will also be undone. In this case, we can recover to the point in time just before the delete statement was issued against the EMPLOYEE, which in our case is T2.

4. Restore the database to the last backup image which we have taken from exercise 3.2.2 at T1:
5. Now that the database is restored, roll forward to a point in time before the delete on table EMPLOYEE was issued which is T2.

```
db2 rollforward db sample to <T2> using local time
```

Note that the timestamp for roll forward has to be provided in this format: `yyyy-mm-dd-hh.mm.ss`.

6. Lastly, take the database out of the roll forward pending status by executing:

```
db2 rollforward database sample stop
```

7. Connect to the sample database and check the number of rows of the STAFF table and the EMPLOYEE table.

```
db2 connect to sample
db2 "select count(*) from staff"
db2 "select count(*) from employee"
```

You will notice that the number of rows returned from the STAFF table is 31 and the number of rows in the EMPLOYEE table is 42.

The “accidentally” deleted rows from the EMPLOYEE table have been recovered by performing a point in time recovery. Roll forward was done up to a time before the delete statement was issued. The delete statement was issued after this point in time; therefore, it was not replayed.

If an END OF LOGS roll forward was done in this case, it would have also replayed the delete statement of the EMPLOYEE table, thereby deleting the rows again. The END OF LOGS option is useful when the database has been lost, and a recovery is needed through all available logs to ensure that all transactions have been recovered.

## 4.3 Scenario 3 – Incremental Backup and Restore

### 4.3.1 Incremental Backup

As database sizes grow larger, it can be quite costly to run full backups, both in terms of storage for the backup images and time required to execute the backups. This is where incremental backups come in. They allow the user to only backup the changes that have been made since the last backup, instead of having to backup the entire database every time.

In order to use incremental backups, the database has to be enabled for it. This is done by turning the `TRACKMOD` database configuration parameter on. When `TRACKMOD` is turned on, the database keeps track of table spaces that have been modified. When an
incremental backup command is issued, it will skip the table spaces that have not been modified since the last backup.

1. Turn the TRACKMOD database configuration parameter to ON:

   ```
   db2 connect to sample
   db2 update db cfg for sample using TRACKMOD ON
   ```

   A warning message will be returned:

   ```
   SQL1363W  One or more of the parameters submitted for immediate modification were not changed dynamically. For these configuration parameters, all applications must disconnect from this database before the changes become effective.
   ```

2. In order for the change of configuration to take effect, reconnect to it.

   ```
   db2 terminate
   db2 connect to sample
   ```

3. Incremental backups require a full backup to act as a reference point for incremental changes. Create a backup of the database using the online mode:

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

   Write down the timestamp of this backup, it will be referred to as T3.

   T3:

4. Make some changes to the STAFF table by decreasing the salary of everyone:

   ```
   db2 connect to sample
   db2 "update staff set salary=salary*0.9"
   ```

5. After the database has enabled incremental backups by modifying TRACKMOD to ON and after creating a full backup of the database, an incremental backup can be now taken to just include the changes made.

   ```
   db2 backup db sample incremental to /home/db2inst1/backups
   ```

   Note down the timestamp at which the incremental backup is created. This will be referred to as T4.

   T4:

6. Compare the size of the full backup and the incremental backup images. At the command prompt, run the following command to check the size:

   ```
   ls -lrt /home/db2inst1/backups
   ```
The circle indicates the size of the last two backup images. Notice the size of the last image (the incremental backup image) is much smaller than the image above it (the full backup image). This is because the incremental image contains only the changes since last full backup. Any table space that was not modified since the last full backup will not be included in the incremental database backup.

### 4.3.2 Incremental Restore

When restoring from incremental backups, the right sequence of full, incremental and incremental delta backups have to be applied. This can become very complex very quickly in a real environment. For this reason, there is an **AUTOMATIC** option available with the restore command such that DB2 figures out the right sequence for applying backups and then applies them. There is also a **MANUAL** option available, but the AUTOMATIC option is highly recommended.

The `db2ckrst` utility can be used to query the database history and generate a list of backup image time stamps needed for an incremental restore.

```
db2ckrst -d sample -t <T4>
```

This output shows that last incremental image will be read first to get the control and header information only. Then the database will be restored from the full backup image. Lastly, the incremental image will be read again, this time applying the data in the image.

Issue the following command from the command line to restore SAMPLE database to the last incremental backup image:
db2 "restore db sample incremental automatic from 
/home/db2inst1/backups taken at <T4>"