CHAPTER 6

High Availability

Ten and one-half percent (10.5%) of the DB2 9 for Linux, UNIX, and Windows Database Administration Upgrade exam (Exam 736) is designed to evaluate your knowledge of transactions and transaction logging and to test your ability to back up and restore a database and to successfully establish a high availability disaster recovery (HADR) environment. The questions that make up this portion of the exam are intended to evaluate the following:

- Your ability to use the RECOVER DATABASE command
- Your knowledge of high availability disaster recovery (HADR)

This chapter is designed to introduce you to the backup and recovery tools that are available with DB2 and to show you how to both restore a damaged database with the RECOVER DATABASE command and how to set up an HADR environment.

Transactions and Transaction Logging

A transaction (also known as a unit of work) is a sequence of one or more SQL operations grouped together as a single unit, usually within an application process. The initiation and termination of a single transaction defines points of data consistency within a database; either the effects of all operations performed within a transaction are applied to the database and made permanent (committed), or the effects of all operations performed are backed out (rolled back), and the database is returned to the state it was in before the transaction was initiated.

In most cases, transactions are initiated the first time an executable SQL statement is executed after a connection to a database has been made or immediately after a preexisting transaction has been terminated. Once initiated, transactions can be implicitly terminated using a feature known as "automatic commit" (in which case, each executable SQL statement is treated as a single transaction, and any changes made by that statement are applied to the database if the statement executes successfully or are discarded if the statement fails), or they can be explicitly terminated by executing the COMMIT or the ROLLBACK SQL statement. The basic syntax for these two statements is:

```
COMMIT <WORK>
```

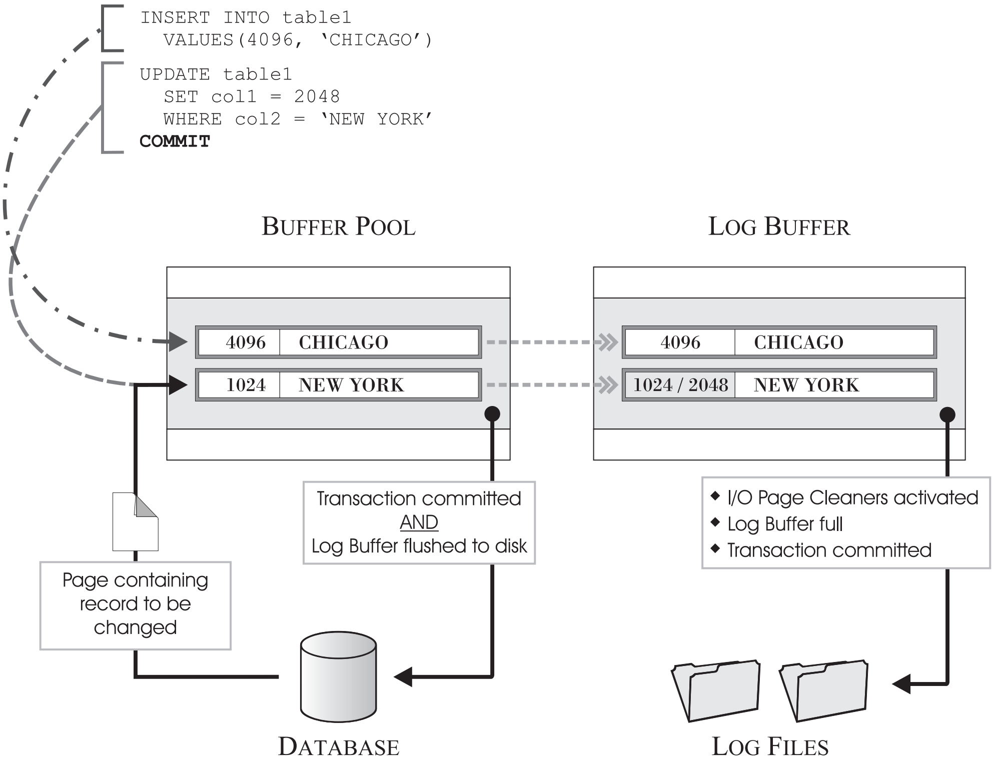
and

```
ROLLBACK <WORK>
```

When the COMMIT statement is used to terminate a transaction, all changes made to the database since the transaction began are made permanent. On the other hand, when the ROLLBACK statement is used, all changes made are backed out, and the database is returned to the state it was in just before the transaction began.

Transaction logging is simply a process used to keep track of changes made to a database (by a transaction), *as they occur*. Each time an update or a delete operation is performed, the page containing the record to be updated/deleted is retrieved from storage and copied to the appropriate buffer pool, where it is then modified by the update/delete operation. (If a new record is created by an insert operation, that record is created directly in the appropriate buffer pool.) Once the record has been modified (or inserted), a record reflecting the modification/insertion is written to the log buffer, which is simply another designated storage area in memory. If an insert operation is performed, a record containing the new row is written to the log buffer; if a delete operation is performed, a record containing the row's original values is written to the log buffer; and if an update operation is performed, a record containing the row's original values, combined with the row's new values, is written to the log buffer. (If replication has not been enabled, an Exclusive OR operation is performed using the "before" and "after" rows and the results are written to the log buffer.)

Whenever buffer pool I/O page cleaners are activated, the log buffer becomes full, or a transaction is terminated (by being committed or rolled back), all records stored in the log buffer are immediately written to one or more log files stored on disk. As soon as all log records associated with a particular transaction have been externalized to one or more log files, the effects of the transaction itself are recorded in the database (i.e., executed against the appropriate table space containers for permanent storage). The modified data pages remain in memory, where they can be quickly accessed if necessary—eventually, they will be overwritten as newer pages are retrieved from storage. The transaction logging process can be seen in Figure 6–1.



```
INSERT INTO table1
  VALUES(4096, 'CHICAGO')

UPDATE table1
  SET col1 = 2048
  WHERE col2 = 'NEW YORK'
COMMIT
```

Figure 6–1: The transaction logging process.

Since log records are externalized frequently and because changes made by a particular transaction are only externalized to the database when the transaction itself is successfully terminated, the ability to return a database to a consistent state after a failure occurs is guaranteed—when the database is restarted, log records are analyzed, and each record that has a corresponding COMMIT record is reapplied to the database; every record that does not have a corresponding COMMIT record is either ignored or backed out (which is why "before" and "after" information is recorded for all update operations).

Database Recovery Concepts

Over time, a database can encounter any number of problems, including power interruptions, storage media failure, and application abends. All of these can result in database failure, and each failure scenario requires a different recovery action.

The concept of backing up a database is the same as that of backing up any other set of data files: you make a copy of the data and store it on a different medium where it can be accessed in the event the original becomes damaged or destroyed. The simplest way to back up a database is to shut it down to ensure that no further transactions are processed and then back it up using the Backup utility provided with DB2. Once a backup image has been created, you can use it to rebuild the database later if for some reason it becomes damaged or corrupted.

The process of rebuilding a database is known as recovery, and three types of recovery are available with DB2:

- Crash recovery
- Version recovery
- Roll-forward recovery

Crash Recovery

Crash recovery is performed by using information stored in the transaction log files to complete any committed transactions that were in memory (but had not yet been externalized to storage) when the transaction failure occurred, roll back any incomplete transactions found, and purge any uncommitted transactions from memory. Once a database is returned to a consistent and usable state, it has attained what is known as a "point of consistency." Crash recovery is illustrated in Figure 6–2.

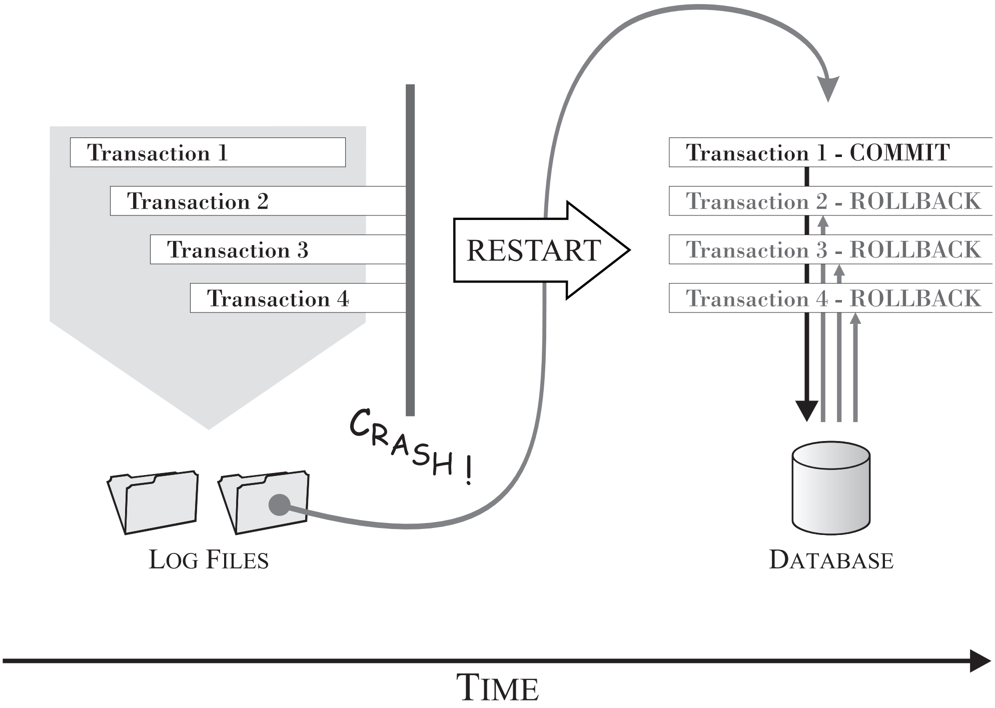


Figure 6–2: Crash recovery.

A crash recovery operation is initiated by executing the RESTART DATABASE command.

Version Recovery

Version recovery is the process used to return a database to the state it was in at the time a particular backup image was made. Version recovery is performed by replacing the current version of a database with a previous version, using a copy that was made with a backup operation—the entire database is rebuilt using a backup image that was created earlier. Unfortunately, when a version recovery is performed, all changes made to the database since the backup image used was created are lost. Version recovery is illustrated in Figure 6–3.

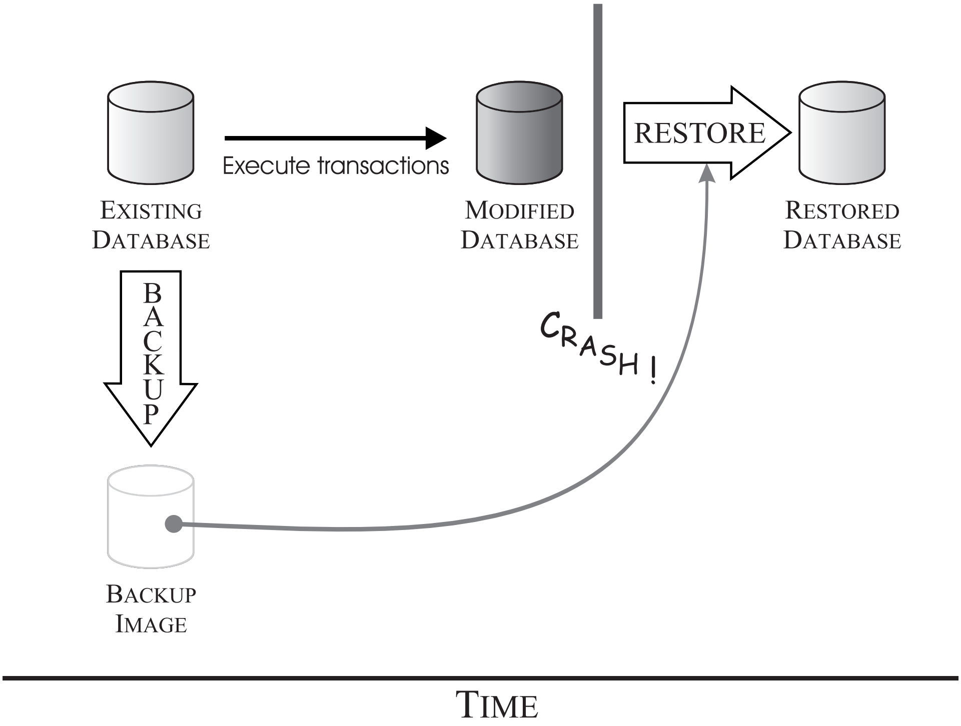


Figure 6–3: Version recovery.

A version recovery operation is initiated by executing the RESTORE DATABASE command; database backup images needed for version recovery operations are generated by executing the BACKUP DATABASE command.

Roll-Forward Recovery

Roll-forward recovery takes version recovery one step farther by rebuilding a database or one or more individual table spaces using a backup image and replaying information stored in transaction log files to return the database/table spaces to the state they were in at an exact point in time. In order to perform a roll-forward recovery operation, you must have archival logging enabled, you must have either a full backup image of the database or a complete set of table space backup images available, and you must have access to all archived log files that have been created since the backup image(s) were made. Roll-forward recovery is illustrated in Figure 6–4.

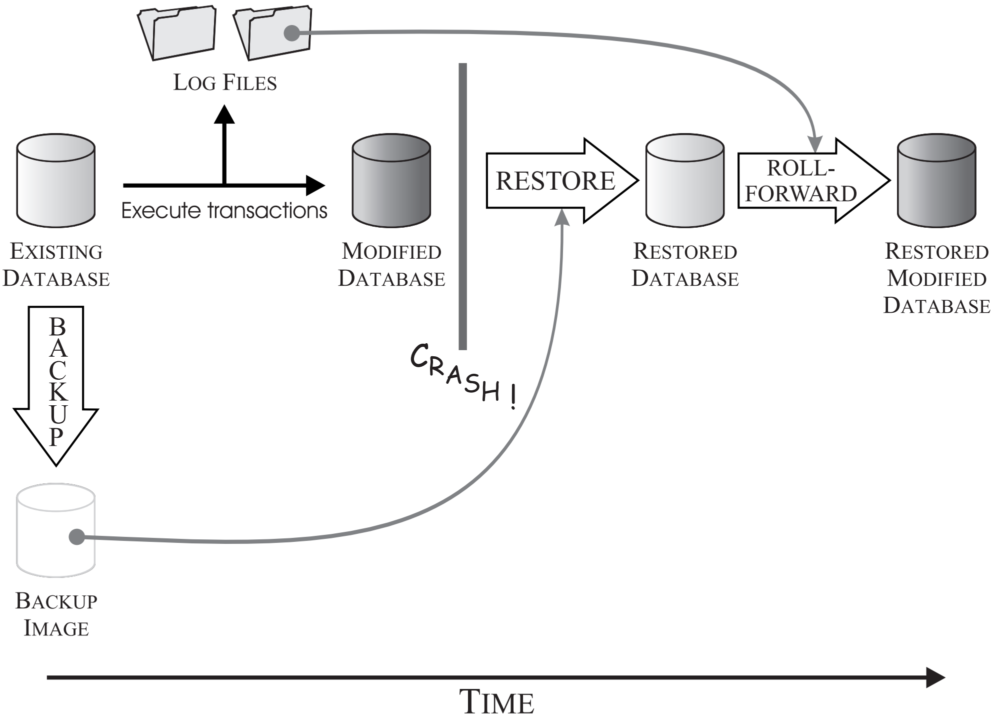


Figure 6–4: Roll-forward recovery.

A roll-forward recovery operation is initiated by executing the ROLLFORWARD DATABASE command.

A Word About the Recovery History File

When a new DB2 database is created, a special file known as the recovery history file is built as part of the database creation process. This file is used to log historical information about specific actions that are performed against the database with which the file is associated. Specifically, records are written to the recovery history file whenever any of the following actions are performed:

- A backup image of any type is created.
- A version recovery operation is performed either on the database or on one of its table spaces.

- A table is loaded, using the Load utility.
- A roll-forward recovery operation is performed either on the database or on one of its table spaces.
- A table space is altered.
- A table space is quiesced.
- Data in a table is reorganized, using the REORG utility.
- Statistics for a table are updated, using the RUNSTATS utility.
- A table is deleted (dropped).

In addition to identifying the event that was performed, each entry in the recovery history file identifies the date and time the event took place, how the event took place, and the table spaces and tables that were affected and, if the action was a backup operation, the location where the backup image produced was stored, along with information on how to access this image. And because the recovery history file contains image location information for each backup image available, it can act as a tracking and verification mechanism during version recovery operations.

The DB2 Recover Utility

While the RESTORE DATABASE command can be used to return a database to the state it was in at the time a backup image was made, and the ROLLFORWARD DATABASE command can be used to replay information recorded in a database's transaction log files to return a database to the state it was in at a specific point in time, if you have a current recovery history file, you can perform both operations in a single step using the Recover utility.

Introduced in DB2 9, the Recover utility performs the necessary restore and roll-forward operations to recover a database to a specific point in time, based on information found in the recovery history file. The Recovery utility is invoked by executing the RECOVER DATABASE command. The basic syntax for this command is:

```
RECOVER [DATABASE | DB] [DatabaseAlias]
<TO [PointInTime] <USING [UTC | LOCAL] TIME>>
<ON ALL DBPARTITIONNUMS>
<USER [UserName] <USING [Password]>>
```

```
   <USING HISTORY FILE ([HistoryFile])>
   <OVERFLOW LOG PATH ([LogDirectory] ,...)>
   <RESTART>
```

or

```
   RECOVER [DATABASE | DB] [DatabaseAlias]
   <TO END OF LOGS
        <ON ALL DBPARTITIONNUMS |
         ON DBPARTITIONNUM<S> ([PartitionNum],...)>>
   <USER [UserName] <USING [Password]>>
   <USING HISTORY FILE ([HistoryFile])>
   <OVERFLOW LOG PATH ([LogDirectory] ,...)>
   <RESTART>
```

where:

DatabaseAlias Identifies the alias assigned to the database associated with the backup image that is to be used to perform a version recovery operation.

PointInTime Identifies a specific point in time, identified by a timestamp value in the form *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds), to which the database is to be rolled forward. (Only transactions that took place before and up to the date and time specified will be reapplied to the database.)

PartitionNum Identifies, by number, one or more database partitions (identified in the db2nodes.cfg file) that transactions are to be rolled forward on. In a partitioned database environment, the Recover utility must be invoked from the catalog partition of the database.

UserName Identifies the name assigned to a specific user under whom the recovery operation is to be performed.

Password Identifies the password that corresponds to the name of the user under whom the recovery operation is to be performed.

HistoryFile Identifies the name assigned to the recovery history log file that is to be used by the Recovery utility.

LogDirectory Identifies the directory that contains offline archived log files that are to be used to perform the roll-forward portion of the recovery operation.

Thus, if you wanted to perform a full recovery operation on a database named SAMPLE (which already exists) using information stored in the recovery history file, you could do so by executing a RECOVER DATABASE command that looks something like this:

```
RECOVER DATABASE sample
TO END OF LOGS
```

On the other hand, if you wanted to restore a database named SAMPLE and roll it forward to an extremely old point in time that is no longer contained in the current recovery history file, you could do so by executing a RECOVER DATABASE command that looks something like this (assuming you have a copy of an older recovery history file available):

```
RECOVER DATABASE sample
TO 2005-01-31-04.00.00
USING HISTORY FILE (/home/user/old2005files/db2rhist.asc)
```

It is important to note that if the Recover utility successfully restores a database, but for some reason fails while attempting to roll it forward, the 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, you need to execute the RECOVER DATABASE command with the RESTART option specified. There is no way to explicitly restart a recovery operation from a point of failure.

High Availability Disaster Recovery (HADR)

High availability disaster recovery (HADR) is a DB2 database replication feature that provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary, to a target database, called the standby. In an HADR environment, applications can only access the current primary database—synchronization with the standby database occurs by rolling forward transaction log data that is generated on the primary database and shipped to the standby database. And with HADR, you can choose the level of protection you want from potential loss of data by specifying one of three synchronization modes: synchronous, near synchronous, or asynchronous.

HADR is designed to minimize the impact to a database system when a partial or a complete site failure occurs. A partial site failure can be caused by a hardware, network, or software (DB2 or operating system) malfunction. Without HADR, a partial site failure requires restarting the server and the instance where one or more DB2 databases reside. The length of time it takes to restart the server and the instance is unpredictable; if the transaction load was heavy at the time of the partial site failure, it can take several minutes before a database is returned to a consistent state and made available for use. With HADR, the standby database can take over in seconds. Furthermore, you can redirect the clients that were using the original primary database to the standby database (which is now the new primary database) by using automatic client reroute or retry logic in the applications that interact with the database. After the failed original primary server is repaired, it can rejoin the HADR pair as a standby database if both copies of the database can be made consistent. And once the original primary database is reintegrated into the HADR pair as the standby database, you can switch the roles so that the original primary database once again functions as the primary database. (This is known as failback operation.)

A complete site failure can occur when a disaster, such as a fire, causes the entire site to be destroyed. Because HADR uses TCP/IP to communicate between a primary and a standby database, the databases can reside it two different locations. For example, your primary database might be located at your head office in one city, whereas your standby database is located at your sales office in another city. If a disaster occurs at the primary site, data availability is maintained by having the remote standby database take over as the primary database.

Requirements for HADR Environments

To achieve optimal performance with HADR, the system hosting the standby database should consist of the same hardware and software as the system where the primary database resides. If the system hosting the standby database has fewer resources than the system hosting the primary database, the standby database may not be able to keep up with the transaction load generated by the primary database.

This can cause the standby database to fall behind or the performance of the primary database to suffer. But more importantly, if a failover situation occurs, the new primary database may not have the resources needed to adequately service the client applications. And because buffer pool operations performed on the primary database are replayed on the standby database, it is important that the primary and standby database servers have the same amount of memory.

IBM recommends that you use identical host computers for the HADR primary and standby databases. (If possible, they should be manufactured by the same vendor and have the same architecture.) Furthermore, the operating system on the primary and standby database servers should be the same version, including patch level. You can violate this rule for a short time during a rolling upgrade, but use extreme caution when doing so. A TCP/IP interface must also be available between the HADR host machines, and a high-speed, high-capacity network should be used to connect the two.

The DB2 software installed on both the primary and the standby database server must have the same bit size (32 or 64), and the version of DB2 used for the primary and standby databases must be identical; for example, both must be either version 8 or version 9. During rolling upgrades, the modification level (for example, the fix pack level) of the database system for the standby database can be later than that of the primary database for a short while. However, you should not keep this configuration for an extended period of time. The primary and standby databases will not connect to each other if the modification level of the database system for the primary database is later than that of the standby database. Therefore, fix packs must always be applied to the standby database system first.

Both the primary and the standby database must be a single-partition database, and they both must have the same database name; however, they do not have to be stored on the same database path. The amount of storage space allocated for transaction log files should also be the same on both the primary and the standby database server; the use of raw devices for transaction logging is not supported. (Archival logging is performed only by the current primary database.)

Table space properties such as table space name, table space type (DMS, SMS, or Automatic Storage), table space page size, table space size, container path, container size, and container type (raw device, file, or directory) must be identical on the primary and standby databases. When you issue a table space statement such as

CREATE TABLESPACE, ALTER TABLESPACE, or DROP TABLESPACE on the primary database, it is replayed on the standby database. Therefore, you must ensure that the table space containers involved with such statements exist on both systems before you issue the table space statement on the primary database. (If you create a table space on the primary database, and log replay fails on the standby database because the containers are not available, the primary database does not receive an error message stating that the log replay failed.) Automatic storage databases are fully supported, including replication of ALTER DATABASE statements. Similar to table space containers, the storage paths specified must exist on both the primary and the standby server.

Once an HADR environment has been established, the following restrictions apply:

- Reads on the standby database are not supported; clients cannot connect to the standby database.
- Self Tuning Memory Manager (STMM) can be run only on the current primary database.
- Backup operations cannot be performed on the standby database.
- Redirected restore is not supported. That is, HADR does not support redirecting table space containers. However, database directory and log directory changes are supported.
- Load operations with the COPY NO option specified are not supported.

Setting Up an HADR Environment

The process of setting up an HADR environment is fairly straightforward. After ensuring that the systems to be used as primary and secondary server are identical and that a TCP/IP connection exists between them, you simply perform the following tasks, in the order shown:

1. Determine the host name, host IP address, and the service name or port number for both the primary and the secondary database server.

   If a server has multiple network interfaces, ensure that the HADR host name or IP address maps to the intended interface. You will need to allocate separate HADR ports for each protected database—these cannot be the same as the ports that have been allocated to the instance. The host name can map to only one IP address.

2. Create the standby database by restoring a backup image or initializing a split mirror copy of the database that is to serve as the primary database.

   It is recommended that you do not issue the ROLLFORWARD DATABASE command on the standby database after the restore operation or split mirror initialization. The results of performing a roll-forward recovery operation might differ slightly from replaying the logs on the standby database using HADR. If the primary and standby databases are not identical when HADR is started, an error will occur.

   When setting up the standby database using the RESTORE DATABASE command, it is recommended that the REPLACE HISTORY FILE option be used; use of the following options should be avoided: TABLESPACE, INTO, REDIRECT, and WITHOUT ROLLING FORWARD.

3. Set the HADR configuration parameters on both the primary and the standby databases.

   After the standby database has been created, but before HADR is started, the HADR configuration parameters shown in Table 7.2 need to be set.

Table 7.2 HADR-Specific Database Configuration Parameters

Parameter	Value Range / Default	Description
hadr_db_role	N/A	Read-only. Indicates the current role of the database, if it is part of a high availability disaster recovery (HADR) environment. Valid values are STANDARD, PRIMARY, or STANDBY.
hadr_local_host	Any valid character string Default: NULL	Specifies the local host for high availability disaster recovery (HADR) TCP communication. Either a host name or an IP address can be used.
hadr_local_svc	Any valid character string Default: NULL	Specifies the TCP service name or port number for which the local high availability disaster recovery (HADR) process accepts connections.
hadr_remote_host	Any valid character string Default: NULL	Specifies the TCP/IP host name or IP address of the remote high availability disaster recovery (HADR) node.
hadr_remote_inst	Any valid character string Default: NULL	Specifies the instance name of the remote server. Administration tools, such as the Control Center, use this parameter to contact the remote server. High availability disaster recovery (HADR) also checks whether a remote database requesting a connection belongs to the declared remote instance.
hadr_remote_svc	Any valid character string Default: NULL	Specifies the TCP service name or port number that will be used by the remote high availability disaster recovery (HADR) node.
hadr_syncmode	SYNC, NEARSYNC, ASYNC Default: NEARSYNC	Specifies the synchronization mode to use for high availability disaster recovery (HADR). This determines how primary log writes are synchronized with the standby database when the systems are in peer state. Valid values for this configuration parameter are SYNC (this mode provides the greatest protection against transaction loss, but at a higher cost of transaction response time), NEARSYNC (this mode provides somewhat less protection against transaction loss, in exchange for a shorter transaction response time than that of SYNC mode), and ASYNC (this mode has the highest probability of transaction loss in the event of primary failure, in exchange for the shortest transaction response time among the three modes).
hadr_timeout	1–4,294,967,295 Default: 120	Specifies the time (in seconds) that the high availability disaster recovery (HADR) process waits before considering a communication attempt to have failed.

4. Connect to the standby instance and start HADR on the standby database.

   HADR is started by executing the START HADR command. The basic syntax for this command is:

   ```
   START HADR ON [DATABASE | DB] [DatabaseAlias]
   <USER [UserName] <USING [Password]>>
   AS [PRIMARY <BY FORCE> | SECONDARY]
   ```

   where:

   DatabaseAlias Identifies the alias assigned to the database for which HADR is to be started.

   UserName Identifies the name assigned to a specific user under whom HADR is to be started.

   Password Identifies the password that corresponds to the name of the user under whom HADR is to be started.

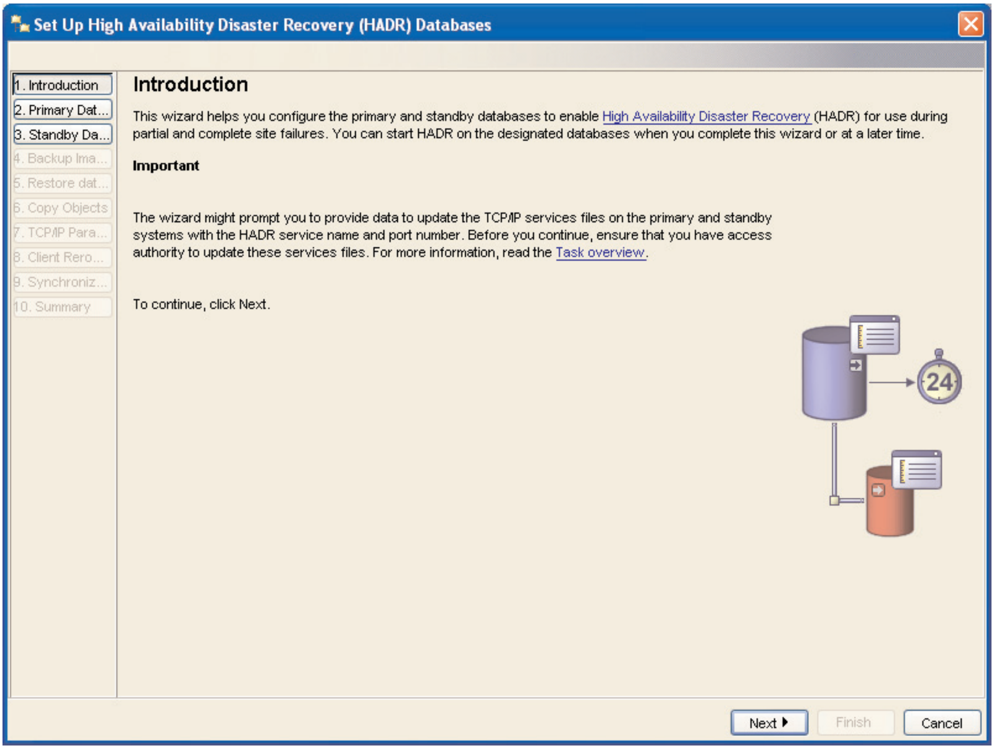
   Thus, if you wanted to start HADR on a database named SAMPLE and indicate that it is to act as a standby database, you could do so by executing a START HADR command that looks something like this:

   ```
   START HADR ON DATABASE sample AS STANDBY
   ```

5. Connect to the primary instance and start HADR on the primary database.In this case, you would execute a START HADR command that looks something like this:

   ```
   START HADR ON DATABASE sample AS PRIMARY
   ```

   You can also set up an HADR environment using the Set Up HADR Databases Wizard, which can be activated by selecting the High Availability Disaster Recovery action from the Databases menu found in the Control Center. Figure 6–5 shows how the first page of the Set Up HADR Databases Wizard might look immediately after it is activated.



Introduction

This wizard helps you configure the primary and standby databases to enable High Availability Disaster Recovery (HADR) for use during partial and complete site failures. You can start HADR on the designated databases when you complete this wizard or at a later time.

Important

The wizard might prompt you to provide data to update the TCP/IP services files on the primary and standby systems with the HADR service name and port number. Before you continue, ensure that you have access authority to update these services files. For more information, read the Task overview.

To continue, click Next.

Figure 6–5: The first page of the Set Up HADR Databases Wizard.

Once an HADR environment has been established, the following operations will be replicated automatically in the standby database whenever they are performed on the primary database:

- Execution of Data Definition Language (DDL) statements (CREATE, ALTER, DROP)
- Execution of Data Manipulation Language (DML) statements (INSERT, UPDATE, DELETE)
- Buffer pool operations
- Table space operations (as well as storage-related operations performed on automatic storage databases)
- Online reorganization

- Offline reorganization
- Changes to metadata for stored procedures and user-defined functions (UDFs)

HADR does not replicate stored procedure and UDF object and library files. If this type of replication is needed, you must physically create the files on identical paths on both the primary and standby databases. (If the standby database cannot find the referenced object or library file, the stored procedure or UDF invocation will fail on the standby database.)

Non-logged operations, such as changes to database configuration parameters and to the recovery history file, are not replicated to the standby database.

Automatic Client Reroute and HADR

Automatic client reroute is a DB2 feature that allows client applications to recover from a loss of communication with the server so that the application can continue its work with minimal interruption. (If automatic client reroute is not enabled, client applications will receive an error message indicating that a connect attempt has failed due to a timeout and no further attempts will be made to establish a connection with the server.) However, rerouting is possible only when an alternate database location has been specified at the server and the TCP/IP protocol is used.

The automatic client reroute feature can be used with HADR to make client applications connect to the new primary database immediately after a takeover operation. In fact, if you set up HADR using the Set Up High Availability Disaster Recovery (HADR) Databases Wizard, automatic client reroute is enabled by default. If you set up HADR manually, you can enable the automatic client reroute feature by executing the UPDATE ALTERNATE SERVER FOR DATABASE command; automatic client reroute does not use the values stored in the *hadr_remote_host* and *hadr_remote_svc* database configuration parameters.

For example, suppose you have cataloged a database named SALES on a client workstation as being located at host named SVR1. Database SALES is the primary database in an HADR environment and its corresponding standby database, also named SALES, resides on host named SVR2 and listens on port number 456. To enable automatic client reroute, you simply specify an alternate server for the SALES database stored on host SVR1 by executing the following command:

```
UPDATE ALTERNATE SERVER FOR DATABASE sales
USING HOSTNAME svr2 PORT 456
```

Once this command is executed, the client must connect to host SVR1 to obtain the alternate server information. Then, if a communication error occurs between the client and the SALES database at host SVR1, the client will first attempt to reconnect to the SALES database on host SVR1. If this fails, the client will then attempt to establish a connection with the standby SALES database located on host SVR2.

Practice Questions

Question 1

Which of the following is NOT a valid statement about the RECOVER DATABASE command?

❍ A. The RECOVER DATABASE command performs the necessary restore and roll-forward operations to recover a database.

❍ B. The RECOVER DATABASE command only rolls a database forward to the end of logs; it cannot roll forward to a specific point in time.

❍ C. The RECOVER DATABASE command can only be used successfully if the recovery history file for the database is available.

❍ D. The RECOVER DATABASE command cannot continue a previously unsuccessful recover operation from the point of failure; if a failure occurs, the recovery operation must be performed again.

Question 2

Which of the following statements is NOT true regarding the use of the RECOVER DATABASE command in a partitioned database environment?

❍ A. If the recovery operation is to a specific point in time, it affects all partitions found in the db2nodes.cfg file.

❍ B. If the recovery operation is to the end of logs, it only affects the partitions that are specified with the RECOVER DATABASE command.

❍ C. The RECOVER DATABASE command can be invoked from any database partition provided it is prefixed with db2_all.

❍ D. The RECOVER DATABASE command can only be used successfully if the recovery history file for the database is available.

Question 3

Given two servers named SVR1 and SVR2 with a database named SALES on SRV1, in what order should the following steps be performed to set up an HADR environment using SRV2 as a standby server?

a) Backup the SALES database on SVR1.

b) Determine the host name, host IP address, and the service name or port number for SVR1 and SVR2.

c) Start HADR on SVR2.

d) Set the HADR configuration parameters on SVR1 and SVR2.

e) Restore the SALES database on SVR2.

f) Start HADR on SVR1.

❍ A. b, a, e, d, f, c

❍ B. b, d, f, c, a, e

❍ C. b, a, e, d, c, f

❍ D. f, c, b, d, a, e

Question 4

Which of the following is NOT an operation that is replicated in an HADR environment?

❍ A. Database configuration changes

❍ B. Execution of Data Definition Language (DDL) statements

❍ C. Online table reorganizations

❍ D. Execution of Data Manipulation Language (DML) statements

Question 5

Which of the following is NOT supported in an HADR environment?

❍ A. Automatic client reroute

❍ B. Automatic storage databases

❍ C. Write operations to the standby database

❍ D. Single partitioned databases

Question 6

Which of the following is NOT a requirement for an HADR environment?

❍ A. The operating system on the primary server and the standby server must be the same (including fix pack level).

❍ B. The database path on the primary server and the standby server must be the same.

❍ C. The DB2 software version and bit size (32 or 64) used on the primary server and the standby server must be the same.

❍ D. Table spaces and table space containers on the primary server and the standby server must be identical.

Answers

Question 1

The correct answer is **B**. The Recover utility performs the necessary restore and roll-forward operations to recover a database to a specific point in time, based on information found in the recovery history file. (The Recovery utility is invoked by executing the RECOVER DATABASE command.) If the Recover utility successfully restores a database, but for some reason fails while attempting to roll it forward, the entire recovery operation must be performed again. There is no way to restart a recovery operation from a point of failure.

Question 2

The correct answer is **C**. In a partitioned database environment, the Recover utility must be invoked from the catalog partition of the database.

Question 3

The correct answer is **C**. After ensuring the systems to be used as primary and secondary server are identical and that a TCP/IP connection exists between them, you can establish an HADR environment by performing the following tasks, in the order shown:

1. Determine the host name, host IP address, and the service name or port number for both the primary and the secondary database server.
2. Create the standby database by restoring a backup image or initializing a split mirror copy of the database that is to serve as the primary database.
3. Set the HADR configuration parameters on both the primary and the standby databases.
4. Connect to the standby instance and start HADR on the standby database.
5. Connect to the primary instance and start HADR on the primary database.

Question 4

The correct answer is **A**. Once an HADR environment has been established, the following operations will be replicated automatically in the standby database whenever they are performed on the primary database:

- Execution of Data Definition Language (DDL) statements (CREATE, ALTER, DROP)
- Execution of Data Manipulation Language (DML) statements (INSERT, UPDATE, DELETE)
- Buffer pool operations
- Table space operations
- Online reorganization
- Offline reorganization
- Changes to metadata for stored procedures and user-defined functions (UDFs)

HADR does not replicate stored procedure and UDF object and library files. If this type of replication is needed, you must physically create the files on identical paths on both the primary and standby databases. (If the standby database cannot find the referenced object or library file, the stored procedure or UDF invocation will fail on the standby database.)

Non-logged operations, such as changes to database configuration parameters and to the recovery history file, are not replicated to the standby database.

Question 5

The correct answer is **C**. In an HADR environment, applications can only access the current primary database—synchronization with the standby database occurs by rolling forward transaction log data that is generated on the primary database and shipped to the standby database.

Question 6

The correct answer is **B**. Both the primary and the standby database must be a single-partition database and they both must have the same database name; however, they do not have to be stored on the same database path.