This tutorial highlights the data integrity skills you need in order to protect your database against unexpected failures. Learn how to configure and manage the high availability (HA) features of DB2® V10.1, which introduced the HADR multiple standby setup that provides a true HA and disaster recovery (DR) solution for your mission-critical databases. Examples illustrate how to configure this feature. You also learn about the DB2 pureScale® technology that provides continuous HA to your critical business operations. This is part six of a series of eight DB2 10.1 DBA for Linux®, UNIX®, and Windows® certification exam 611 tutorials.

View more content in this series

Before you start

Learn what to expect from this tutorial and how to get the most out of it.

About this series

The DB2 10.1 DBA for Linux, UNIX, and Windows certification exam 611 prep series covers the topics you need to know to take the test. Taking the exam showcases your proficiency on the topics and gives you exposure to detailed information on using the new features available with each version. Whether you take the exam now or later, preparing for it can be a great job aid.

About this tutorial

About 10% of the DB2 10.1 DBA for Linux, UNIX, and Windows exam 611 concerns your ability to administer the database backup and recovery operations, at both the database and table space level, and your ability to configure and manage DB2 HA and DR features with the multiple standby configurations.
This tutorial covers:

- How to perform crash recovery, roll-forward recovery, and version recovery.
- Database level and table-space level backup and recovery operations.
- How to configure and manage HADR (read from standby).
- DB2 pureScale HA features.

**Objectives**

After completing this tutorial, you should be able to:

- Perform the data integrity operations such as crash recovery, roll forward recovery, and version recovery in the event of a failure.
- Perform the different kinds of backup and recovery operations.
- Configure and manage HADR with a multiple standby setup, and know how to enable read on standby.
- Demonstrate knowledge of the DB2 pureScale HA features.

**Prerequisites**

To take the DB2 10.1 for Linux, UNIX, and Windows DBA certification exam 611, you must have already passed the DB2 10.1 Fundamentals exam 610 or the DB2 9 Fundamentals exam 730. This tutorial is just one among the many resources available for you to prepare for the exam (see Resources for a list of other available resources).

For this tutorial, you need a basic knowledge of DB2 products, tools, instances, and databases.

**System requirements**

You do not need to have DB2 installed to follow the contents of the tutorial, but it would help to understand some of the operations described here. Not all of them can be tried with a free trial version of DB2, however.

**Database recovery**

The ability to recover a database in dire situations is a critical component of database administration. For example, if DB2 crashes unexpectedly, it needs to be able to recover and continue processing where it left off. Or, in the case of disk corruption or data corruption, it's necessary to perform a database restore and apply all the changes that occurred in the database up until the time of corruption. There might be a situation where you need to recover a single table space up to a point in time. With natural disasters such as flooding or earthquakes, database recovery plays an important role in a business's ability to recover and continue operations.

There are basically three types of recovery:

- Crash recovery
- Version recovery
• Roll-forward recovery

**Crash recovery**

DB2 has the ability to recover automatically from most types of crashes and will initiate the crash recovery process once a user tries to connect to, or activate, the database after a crash. During this process, the database is moved back to a consistent state by rolling back or completing the transactions that were interrupted when the crash occurred.

The following conditions can cause the database or the database manager to fail:

- A power failure on the server causing the DB2 instance to go down.
- A hardware failure such as memory, disk, CPU, or network.
- An operating system failure on the server.

You need to enable the automatic restart configuration parameter if you want DB2 to perform automatic crash recovery. The default setting is `ON`. If the automatic restart configuration parameter is set to `OFF`, you have to issue the `RESTART DATABASE` command to start the crash recovery process. If the database I/O was suspended during the crash, you need to use the `WRITE RESUME` option of the `RESTART DATABASE` command to complete the crash recovery successfully.

If the database under crash recovery was enabled for roll-forward recovery and if DB2 encounters a damaged table space, that table space is taken offline and DB2 continues to recover other table spaces. The offline table spaces need to be investigated later and recovered. However, if the table space that is offline contains the system catalog tables, then the database remains inaccessible until this table space is recovered. This **does not apply** to the DB2 pureScale environments. If an error occurs during the member crash recovery or group crash recovery, the recovery operation fails.

When doing table space restores in the event of damaged table spaces, the roll forward recovery must be performed through the end of logs. If the table space is a temporary table space, then you can drop and recreate a new temporary table space.

If the database or database manager configuration parameter `INDEXREC` is set to `RESTART`, all invalid indexes have to be rebuilt during database activation. If the parameter is set to `ACCESS`, the indexes will be rebuilt during the first access of the database. If there are incomplete reorganization requests present in the temporary table space, then the `INDEXREC` parameter needs to be changed to `ACCESS` to avoid restart failures.

**Crash recovery in a partitioned database environment**

In a partitioned database environment, recovery from a transaction failure involves both the failed database partition and any other database partition that was participating in the transaction. Two types of database recovery can occur:

- Crash recovery of the failed database partition after the issue has been corrected.
- Database partition failure recovery on the active database partition, which occurs immediately after detection of the failure.
The coordinator agents running on the coordinator partition, which are responsible for distributing the work across the partitions, keep track of which partitions are involved in a transaction. When an application issues a commit, the coordinator agent uses the two-phase commit to perform the commit operation. It first issues a prepare request to all the partition servers involved, then the partition servers respond with one of the following:

- READ_ONLY: No data change happened on this server
- YES: Data changed on this server
- NO: Data change failed on this server

If a NO is received, the transaction is rolled back. If a YES is received, the coordinator partition writes a COMMIT log record and sends the commit request to the involved partitions. After all the servers are committed, an acknowledgement is sent back to the coordinator partition and a FORGET log record is written.

**Transaction recovery on the active database partition server**

If a partitioned database server detects a failure at another partition, all work associated with that failed partitioned server is stopped.

If the active database partition is the coordinator partition for the application and the application was running on the failed database partition, the coordinator agent is interrupted to do the failure recovery. If the agent is in the second phase of commit, then the error SQL0279N is returned. Otherwise, a rollback is issued.

If the failed partition server is the coordinator partition, the agents working for the application on the other active partitions are interrupted to do failure recovery. The transaction is rolled back locally on each partition where it is not in a PREPARED state. On those partitions where the transaction is in a PREPARED state, the transaction is marked as indoubt. In such a case, the coordinator database partition is not aware of this because the coordinator partition is down.

If the application was connected to the failed database partition server, but neither the local database partition nor the failed partition is the coordinator partition, agents working for this application are interrupted. The coordinator partition either sends a ROLLBACK or DISCONNECT to the other database partition servers.

**Transaction recovery on the failed database partition server**

If the database manager ended abnormally because of the failed transaction, then you can use the restart option with the db2start command to restart the database. If the database partition could not be started, then you can use another database partition and issue db2restart to restart the database manager. The database partitions can be left in an inconsistent state when the database manager ends abnormally. Crash recovery is needed for those partitions to bring them to a consistent state. You can accomplish this in two ways:

- Explicitly with the restart database command.
- Implicitly when connecting to the database when the autorestart configuration parameter is set to ON.
Crash recovery reapplies the log records in the active log files to ensure all completed transactions are present in the database. All uncommitted transactions are rolled back except for those marked as indoubt. In a partitioned database environment, there can be two kinds of in-doubt transactions. A transaction is in doubt:

- If it is prepared but not committed on a database partition that is not the coordinator partition for the application.
- On a coordinator partition if it committed but not logged. For example, the FORGET log record is not yet written.

Crash recovery attempts to resolve all in-doubt transactions, but some might not be resolved. For example, if the coordinator partition completes the crash recovery before the other failed database partitions involved in the transaction, the in doubt transaction will not be resolved. This is the expected behavior for crash recovery because it is performed by each affected database partition independently.

In-doubt transactions that were not resolved after a crash recovery can affect other processing because those transactions are still holding the resources. Thus, after a crash, it’s important to ensure that no in-doubt transactions remain. You can use the `LIST INDOUBT TRANSACTIONS` command to query, commit, or roll back the in-doubt transactions.

**Determining the failed database partition server**

The application receives one of the following error codes in the case of a failure:

- SQL0279N: A database partition server involved in a transaction is terminated during COMMIT processing.
- SQL1224N: The failed database partition is the coordinator partition for the application.
- SQL1229N: The failed database partition is not the coordinator partition for the application.

To determine which database partition failed, examine the SQLCA field. It contains the node number of the failure and the administration notification log, which contains the node number of the failed server.

**Disaster recovery**

As the name indicates, this type of recovery is performed when a natural disaster or other catastrophic event occurs. Businesses often plan for this type of contingency in one of the following ways:

- A standby site to be used in the case of a disaster at the primary site.
- A different machine in which to recover the database.
- Off site storage of database and table space backups and the associated archived logs.

You can restore the database on another machine using the most recent backup image or you can keep the database backups and logs at the standby site to be restored in case of a disaster. You can also establish a standby database and keep it in sync by constantly applying logs using log
shipping or other tools, such as HADR or replication, to keep another copy of the database running in a different location.

**Version recovery**

You can restore the database to a prior version using version recovery. However, all the changes applied after that version will not be available. You can use this method with the recoverable or non-recoverable databases. With recoverable databases, you must use the WITHOUT ROLLING FORWARD option on the restore for version recovery.

In a partitioned database environment, the database is spread across many database partitions. The backup image that you use for version recovery must be a version backup taken on all nodes at the same time.

**Roll-forward recovery**

With roll-forward recovery you can recover a database or table space to a point in time using the backup image and the archived logs. To use this method, you must enable the database for roll-forward recovery; specifically, you should not set the logarchmeth1 or logarchmeth2 parameters to OFF. The database or table space backups and the associated logs must be available.

If you use the WITHOUT ROLLING FORWARD option with the restore, it is equivalent to using the version recovery. If you do not use that option, then the database is left in the roll-forward pending state at the end of the restore, which allows the roll-forward recovery to be carried out.

The WITHOUT ROLLING FORWARD option cannot be used if the backup image is an online backup image or a table space backup image. During the roll forward operation, the archived log files are retrieved from the archive device. If the logs are compressed during archive, they are automatically uncompressed during retrieval.

There are two types of roll-forward recovery:

**Database roll-forward recovery**

DB2 records all changes made to the database in the log files. These changes in form are transactions and are applied to the database during the roll-forward operation. You can perform the roll-forward operation to a particular point in time or to the end of logs. In a partitioned database base environment, you must perform the roll forward on the catalog partition. For a point-in-time recovery, all database partitions must be rolled forward to the same point in time. If you are recovering only one database partition, you can use the end of logs option to bring it to the same level as other partitions. With partitioned databases, point-in-time recovery applies to all database partitions.

**Table space roll-forward recovery**

If a database is enabled for roll-forward recovery, it is also enabled for table space level roll-forward recovery. To perform this operation, you need either the full database backup or one or more individual table space backups and the logs needed for roll forward operation. You can roll forward to a point in time or to the end of logs.
You use the table space roll-forward recovery in two kinds of situations:

- After a table space restore, it is left in a roll-forward pending state. It becomes necessary to perform the roll-forward operation to bring it to normal state.

- After a crash recovery, some table spaces can be left in a roll-forward pending state. To correct this issue, it may not be necessary to restore the table space. If the underlying problem with the table space can be corrected, just issuing the `roll forward` command to end of logs brings the table space to a normal state. However, if the table space in question is the catalog table space, then that table space must be restored and rolled forward to make the database usable.

In a partitioned database environment, if a table space is rolling forward to a point in time, it applies to all the database partitions where the table space resides. On the other hand, if you are doing the roll forward to the end of logs and you do not want some of the database partitions to participate in that operation, then you have to specify the list of partitions to be involved in the roll forward.

If a table is a partitioned table and its partitions are in different table spaces, then a point in time recovery should include all the table spaces containing the partitions, including any detached, attached, or dropped partitions. However, if you are rolling forward to end of logs, it can be performed on a single table space.

**Storage group modifications during roll forward recovery**

If there are no storage group redefines done as part of a redirected restore, the roll-forward recovery will replay any storage group changes such as storage path updates, storage group renames, and table space storage group association updates described in the log records. If a storage path is not found during roll forward, the error SQL1051N is returned.

If storage paths were redefined as part of the restore, then the changes to storage groups found in the log records are not applied. However, changes to data tag or storage group names are applied. Any `CREATE STOGROUP` or `DROP STOGROUP` operations found in the log records are replayed with the specified paths.

Any rebalance operations in the log records are replayed during roll forward recovery. The rebalance might not finish at the end of a roll forward operation, in which case it is suspended and resumed at the first activation of the database.

**Performing database backup and recovery operations**

As a database administrator, it's important to develop a database backup and recovery strategy to maintain business continuity and availability in the case of a disaster, data loss, data corruption, and so on. The strategy should ensure all the information required for a database recovery is available when it is needed. The database recovery history file, located in the database directory, contains the summary of backup information that you can use to determine the recovery operations. The table space change history file, also in the database directory, contains the log file information needed to recover a table space.
You can use automatic maintenance to schedule database backups by updating the database configuration value. When this is set, users can run manual backups as well. DB2 performs automatic backups only when they are required.

The frequency and scheduling of the backups determine how long the database recovery takes. Based on the database size and the work load on the database, you can schedule a combination of full and incremental database backups. The frequency of backups is also dependent on the availability of storage space.

**Online and offline backups**

If a database is enabled for roll-forward recovery, you can run online as well as offline backups on this database. You can take offline backups only when there are no connections to the database. On a non-recoverable database, you can take only offline backups.

To restore an online backup, a set of logs spanning the backup operation are also required to complete the restore. Without these logs, it is not possible to recover the database. You have the option of including the logs as part of the backup image itself when you issue the `backup` command to protect against the deletion of these log files. For example:

```
db2 backup database sample online to /path include logs
```

The image contains the logs needed for restoring the database. You can use the `logtarget` option of the `restore` command to extract the log files from the image, as in the following example.

```
db2 restore database sample taken at <timestamp> logtarget /pathname
```

In a partitioned database environment, there are four possible ways to back up the database:

- Back up one partition at a time.
- Use `db2_all` with the `backup database` command to specify all the partitions you want backed up.
- Use the single system view (SSV) backup to back up all or a set of partitions simultaneously.
- Use IBM Data Studio's task assistant to help you through the process.

On a partitioned database environment, SSV is the recommended method. To use SSV, on the catalog database partition, invoke the backup command with the `ONDBPARTITIONNUMS` parameter. With the SSV method, by default, the logs are included in the backup image.

**Incremental backups**

Incremental backups capture only the changes that were made to the database from the last full backup or from the last delta backup. To perform incremental backups, you need to turn the `TRACKMOD` database configuration parameter on to allow DB2 to track the modified pages. This kind of backup is usually scheduled on large databases to conserve storage space without compromising the ability to recover the database. There are two kinds of incremental backups: incremental and incremental delta.
When you specify incremental, DB2 captures all changes made to the database from the time of the last full database backup. If you specify incremental delta, DB2 captures all changes made to the database from the time of the last backup whether it's a full or incremental backup.

For example:

```
db2 backup db sample online incremental use tsm
```

```
db2 backup db sample online incremental delta use tsm
```

Before you schedule an incremental backup, you need to perform a full database backup because an incremental restore requires that the last full database backup be taken before the incremental backup. For large databases, it is not unusual to have a mix of incremental and incremental delta backups done during the week with one full database backup scheduled per week.

**Table space backups**

Separating data into different table spaces, such as placing LOB objects in one table space and indexes in another, protects against recovering everything when a disk failure occurs. You can choose to back up all the table spaces containing part of a table or back up only some table spaces depending on what part of the table changes. However, to perform a point in time recovery, all table spaces that contain a table must be rolled forward to that point.

To know whether a table space changed after the last backup, use the `db2pd -tablespaces trackmodstate` command and the `tbsp_trackmod_state` monitor element.

A better strategy is to back up a table space after a reorg so that the reorg operation need not be replayed during a recovery.

**Snapshot backups**

To perform snapshot backup and restore operations, you need a DB2 ACS API driver for your storage device. For a list of supported storage hardware for the integrated driver, see the Tivoli® documentation for supported storage subsystems (in Resources).

With snapshot backups, individual table spaces cannot be restored, and redirected restores cannot be performed. With a flash copy restore, any changes made to the current storage group paths after the snapshot backup are overwritten by the flash copy restore. For example:

```
db2 backup db sample use snapshot
```

**Split-mirror backups**

Instead of performing a regular backup, you can also take a split-mirror backup image on a database to a different location using a combination of OS level and storage system commands combined with DB2 commands.

To use a split mirror as a backup image:
1. Connect to the database and issue the `write suspend` command, as follows:

   ```
   db2 connect to sample db2 set write suspend for database
   ```

   During this suspend phase, you cannot run any utilities or tools on the database until the split mirror is completed.

2. Create one or more split mirrors on the database by issuing appropriate operating system and storage system commands. Make sure all the database paths are copied, and check the DBPATH admin view of the database to get the list of all paths that comprise the database.

3. After the copy is complete, resume write operations using:

   ```
   db2 set write resume for database
   ```

### Storage space considerations

With version recovery, additional storage space is needed to store the full database backup copies. In the case of roll-forward recovery, storage space is needed to store the database backup image, the table space backup image, and the archive logs needed for restore and roll forward. To save storage space, you can use backup compression and log file compression. To compress database backups, use the `compress` option of the `db2 backup` command. To compress archive logs, use the `logarchcompr1` and `logarchcompr2` database configuration parameters. This is a new capability with DB2 10.1.

Note: The archive log file compression cannot be used when either the `logpath` or the `newlogpath` configuration parameters point to raw devices.

### Backup and restore operations for different operating systems and platforms

The supported backup and restore operations for different platforms can be grouped into one of three families:

- Big-endian Linux and UNIX
- Little-endian Linux and UNIX
- Windows

Table 1 provides details on the endianness of the Linux and UNIX operating systems that DB2 supports.

### Table 1. Endianness of supported Linux and UNIX operating systems

<table>
<thead>
<tr>
<th>Platform</th>
<th>Endianness</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX®</td>
<td>Big endian</td>
</tr>
<tr>
<td>HP on IA64</td>
<td>Big endian</td>
</tr>
<tr>
<td>Solaris x64</td>
<td>Little endian</td>
</tr>
<tr>
<td>Solaris SPARC</td>
<td>Big endian</td>
</tr>
<tr>
<td>Linux on zSeries</td>
<td>Big endian</td>
</tr>
<tr>
<td>Linux on pSeries®</td>
<td>Big endian</td>
</tr>
<tr>
<td>Linux on IA-64</td>
<td>Little endian</td>
</tr>
</tbody>
</table>
Linux on AMD64 and Intel EM64T | Little endian
---|---
32-bit Linux on x86 | Little endian

**Recover database command**

The `recover database` command combines the restore and roll forward operations to recover the database to a point in time or to end of logs based on the information in the recovery history file. The utility finds the best backup image to use and restores that image. The task assistant in IBM Data Studio V3.1 or later can also be used to run the recovery on a database.

The options in Table 2 cannot be used with the `recover database` command.

**Table 2. Options not to be used with recover database command**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLESPACE tablespace-name</td>
<td>Table space restore operations are not supported.</td>
</tr>
<tr>
<td>INCREMENTAL</td>
<td>Incremental restore operations are not supported.</td>
</tr>
<tr>
<td>OPEN num-sessions SESSIONS</td>
<td>You cannot indicate the number of I/O sessions that are to be used with TSM or another vendor product.</td>
</tr>
<tr>
<td>BUFFER buffer-size</td>
<td>You cannot set the size of the buffer used for the restore operation.</td>
</tr>
<tr>
<td>DLREPORT filename</td>
<td>You cannot specify a file name for reporting files that become unlinked.</td>
</tr>
<tr>
<td>WITHOUT ROLLING FORWARD®</td>
<td>You cannot specify that the database is not to be placed in roll forward pending state after a successful restore operation.</td>
</tr>
<tr>
<td>PARALLELISM n</td>
<td>You cannot indicate the degree of parallelism for the restore operation.</td>
</tr>
<tr>
<td>WITHOUT PROMPTING</td>
<td>You cannot specify that a restore operation is to run unattended.</td>
</tr>
</tbody>
</table>

If the `recover database` command is used following a stopped recovery, DB2 attempts to continue the recovery. If you need the entire recovery to be restarted, use the `restart` option of the `recover database` command to start from the beginning. For example:

```bash
db2 recover db sample
```

**Restore database command**

A simple `restore` command requires only the database name or the alias name, as follows:

```bash
db2 restore db sample;
```

This produces a warning during the restore, as shown here:

```
SQL2539W Warning! Restoring to an existing database that is the same as the backup image database. The database files will be deleted. Do you want to continue? (y/n)
```

Responding `y` completes the restore operation. A database restore requires exclusive connection to the database. No other connection is permitted until the restore completes successfully. However, you can do a table space restore online but access to the table space being restored is not allowed until the restore and the roll forward complete on the table space.
When a partial restore is being done, table spaces can be restored from individual backup images or from the full database backup image.

The database path and new storage locations can be defined during a restore operation, as shown in the following examples.

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTORE DATABASE SAMPLE</td>
</tr>
<tr>
<td>RESTORE DATABASE SAMPLE TO /dbdirectory</td>
</tr>
<tr>
<td>RESTORE DATABASE SAMPLE DBPATH ON /dbdirectory</td>
</tr>
<tr>
<td>RESTORE DATABASE SAMPLE ON /path1, /path2, /path3</td>
</tr>
<tr>
<td>RESTORE DATABASE SAMPLE ON E:\newpath1, F:\newpath2 DBPATH ON D:</td>
</tr>
</tbody>
</table>

If TO or DBPATH are specified, the database directory is restored to that location. If a database with the same name already exists, then this value is ignored and the database is restored to the same location. If no DBPATH is specified but paths are specified with the ON clause, then the database directory is restored to the first path in the list. The ON clause specifies the storage paths for the data. Any storage groups defined in the database will use the new paths specified.

You can also use the task assistant in IBM Data Studio to start a restore operation.

Restore from a snapshot backup involves the fast data copy technology of a storage device. To do this, you need a DB2 ACS API driver for your storage device. To restore from a snapshot backup, a snapshot backup needs to be taken. The USE_SNAPSHOT parameter of the restore database command or the db2Restore API with the SQLU_SNAPSHOT_MEDIA media type can be used to perform a snapshot restore. For example:

```
db2 restore db sample use snapshot
```

**Using incremental restore**

After the database has been enabled for incremental backup, you can perform manual or automatic incremental restore.

The following are examples with the corresponding backup commands:

```
db2 backup db sample
Backup successful. Time stamp for the backup image is ts1.
db2 backup db sample incremental.
Backup successful. Time stamp for the backup image is ts2.
```

To restore the db manually, do the following:
restore db sample incremental taken at ts2 without prompting

DB20000I The RESTORE DATABASE command completed successfully.

restore db sample incremental taken at ts1 without prompting

DB20000I The RESTORE DATABASE command completed successfully.

restore db sample incremental taken at ts2 without prompting

DB20000I The RESTORE DATABASE command completed successfully.

The following example shows an automatic incremental restore:

restore db sample automatic incremental taken at ts2 without prompting

DB20000I The RESTORE DATABASE command completed successfully.

**Performing redirected restores**

You can perform a redirected restore in the following situations:

- To restore a backup image to a target machine that is different from the source machine.
- To restore your table space containers into a different physical location.
- If a restore operation failed because one or more containers are inaccessible.
- If you want to redefine the paths of a defined storage group.

There are some restrictions associated with redirected restores. You cannot:

- Use a redirected restore to move data from one operating system to another.
- Create or drop a storage group during the restore process.
- Modify storage group paths during a table space restore process, even if you're restoring all table spaces associated with the storage group.

The redirected restore is a two step process: issuing the `restore` command with the `redirect` option, and an intervening step to define the set of table space containers or storage group paths. At the end, specify `restore database continue`, which makes the new path for the table space containers take effect. During a redirected restore operation, database directory and file containers are created if they don't exist. However, device containers are not created automatically. DB2 provides for adding, changing, and removing table space containers for non-auto storage table spaces and for storage paths for automatic storage table spaces. For SMS table spaces, a redirected restore is the only way to redefine containers.

In a partitioned database environment, you can redirect the storage group paths only from the catalog database partition. Modifying the storage group paths on the catalog partitions places the other partitions in the restore pending state. The catalog partition must be restored before all other partitions. For example:

db2 restore db sample redirect without prompting
SQL1277W A redirected restore operation is being performed.

During a table space restore, only table spaces being restored can have their paths reconfigured. During a database restore, storage group storage paths and DMS table space containers can be reconfigured.

DB20000I The RESTORE DATABASE command completed successfully.

db2 set tablespace containers for 2 using (file '/db2data2', file '/db2data3')

DB20000I The SET TABLESPACE CONTAINERS command completed successfully.

db2 restore db sample continue

DB20000I The RESTORE DATABASE command completed successfully.

To make it easier to perform a redirected restore, you can use the GENERATE SCRIPT option of the command to generate the restore script containing all the set table space container commands.

**High availability disaster recovery (HADR)**

HADR is a powerful replication feature that provides protection against partial and complete site failures. Data changes are constantly replicated from the primary database to the standby database. HADR uses TCP/IP protocol for communication between primary and standby. Therefore, the locations of the primary and standby databases can be different and thus provide a DR solution in case of natural disasters. Figure 1 shows a typical HADR setup.

**Figure 1. HADR setup**

In the case of a failure at the primary site, you can issue a takeover at the standby and thereby make it the primary. After the primary database is recovered, in most cases HADR reconnects to the old primary. The old primary becomes the standby. At this point, you can do another takeover at the old primary site to go back to the original location. This is called the **failback**.
Establishing HADR

Figure 2 shows how to establish HADR between the primary and secondary servers.

**Figure 2. Establishing HADR**

The following synchronization modes are supported by HADR:

**SYNC**
A commit succeeds if the log data is on the disk at both primary and standby.

**NEARSYNC**
A commit succeeds if the log data is on the disk at the primary and written to the main memory on standby.

**ASYNC**
A commit succeeds if the log data is on disk at primary and delivered to standby.

**SUPERASYNC**
Commit on the primary does not wait for acknowledgement.

To establish HADR:

1. Take an offline or online backup on the primary database, restore it on the standby server, and leave it in the roll forward pending state.
2. Choose the SYNC mode of the HADR. The available modes are SYNC, NEARSYNC, ASYNC, and SUPERASYNC.
3. Set up all the HADR configuration database parameters between the primary and the standby.
4. Start HADR on standby.
5. Start HADR on primary.
DB2 V10.1 offers multiple standby mode facilities. You can have up to three standby servers providing a true HA and DR solution for your business. Figure 3 shows what a typical HADR multiple standby setup looks like.

**Figure 3. HADR multiple standby**

HADR V10.1 features

HADR in V10 now supports:

- Multiple standby servers, providing a true HA and DR solution.
- Delayed replay option with HADR, which helps protect against data corruption and application errors.
- HADR log spooling on the standby server, which helps prevent the backup on primary and improves HADR performance.
- Reads on standby are allowed in all HADR states except local catchup state and replay_only mode.
- All the standbys can be used for reads.

With multiple standbys set up, you can have one primary standby server in close sync with the HADR primary server in the same location, and the other two standby servers as auxiliary standbys in a remote location. One of the auxiliary standbys can be set up with delayed log replay to protect against accidental data corruption or data loss on the primary. Auxiliary standbys can only support superasync mode.

The `hadr_peer_wait_limit` and `hadr_peer_window` configuration parameters do not apply to the auxiliary standby servers because they are never in a peer state. You can use the new configuration parameter `hadr_target_list` to specify the host and port names of all the standby servers. The first name specified on the target list will be the principal standby. The principal standby uses the SYNC mode of the primary. The other standbys use the SUPERASYNC mode.
All of the standbys have their local SYNC mode that will be effective if any of them becomes the primary.

IBM Tivoli System Automation for Multiplatforms (SA MP) automated failover is supported only for the principal standby. You must issue a takeover manually on one of the auxiliary standbys to make one of them the primary.

All of the HADR sync modes are supported on the principal standby, but the auxiliary standbys can only be in SUPERASYNC mode.

You can use the HADR_REPLAY_DELAY configuration parameter to specify the delay value, until which time the standby will wait to apply the changes from the primary. There are some restrictions when using this replay delay feature, as follows.

- You can set the `hadr_replay_delay` configuration parameter only on a standby database.
- A `TAKEOVER` command on a standby with replay delay enabled will fail. You must first set the `hadr_replay_delay` configuration parameter to 0, deactivate and reactivate the standby to pick up the new value, then issue the `TAKEOVER` command.
- The delayed replay feature is supported only in SUPERASYNC mode. Because log replay is delayed, a lot of unreplayed log data might accumulate on the standby and fill up the receive buffer and spool (if configured). In other synchronization modes, this would cause the primary to be blocked.

The `hadr_spool_limit` database configuration parameter lets you specify additional space for log spooling on the standby. This parameter sets an upper limit on how much data is spooled to disk if the log receive buffer fills up. Transactions in the primary can progress without waiting for the log replay on standby. It is recommended you use the log spooling feature when the delayed replay is enabled on a standby to be able to receive many more logs than what is limited by the log receive buffer size.

**Requirements for setting up HADR**

- The operating system and the DB2 version and level must be the same on the primary and standby. The only exception to this is when you are performing rolling upgrades.
- The DB2 software for both the primary and the standby database must have the same bit size.
- A TCP/IP interface must be available between the HADR primary and the standby.
- The primary and standby databases must have the same database name.
- Table spaces must be identical on the primary and standby databases.

**Example: Multiple standby setup with read on standby**

Create your standby database or databases by using either a restored backup or split mirror and leave the databases in roll forward pending state.

1. On each of the databases, set the `hadr_local_host`, `hadr_local_svc`, `hadr_local_svc`, and `hadr_sync_mode` configuration parameters, as follows:

   ```sql
   UPDATE DB CFG FOR dbname USING HADR_LOCAL_HOST hostname HADR_LOCAL_SVC servicename HADR_SYNCMODE syncmode
   ```
2. Set the `hadr_target_list` configuration parameter on all of the standbys and the primary.
   
   ```
   DB2 UPDATE DB CFG FOR dbname USING HADR_TARGET_LIST principalhostname:principalservicename |
   auxhostname1:auxservicename1| auxhostname2:auxservicename2
   ```

3. On all the databases, set the `hadr_remote_host`, `hadr_remote_svc`, and `hadr_remote_inst` configuration parameters.
   This step is not required because in multiple standby mode the values are automatically set if you do not set them, and they're automatically reset if you set them incorrectly. However, explicitly setting them to the correct values makes correct values available immediately. These values are helpful for the IBM Tivoli SA MP software, which might require the `hadr_remote_inst` value to construct the resource name.

4. On the primary, set the parameters to the corresponding values on the principal standby by issuing the following command:
   
   ```
   DB2 "UPDATE DB CFG FOR dbname USING HADR_REMOTE_HOST principalhostname HADR_REMOTE_SVC
   principalservicename HADR_REMOTE_INST principalinstname"
   ```

5. On each standby, set the parameters to the corresponding values on the primary by issuing the following command:
   
   ```
   DB2 "UPDATE DB CFG FOR dbname USING HADR_REMOTE_HOST primaryhostname HADR_REMOTE_SVC
   primaryservicename HADR_REMOTE_INST primaryinstname"
   ```

6. To set up read on standby on any of the standby servers, issue the following commands:
   
   ```
   - db2set DB2_HADR_ROS= - db2set DB2_STANDBY_ISO=UR
   ```

7. Connect to each standby instance.

8. On the standby instance, issue the `START HADR` command with the `AS STANDBY` parameter, as follows:
   
   ```
   START HADR ON DB dbname AS STANDBY
   ```

9. Connect to the primary instance.

10. On the primary instance, issue the `START HADR` command with the `AS PRIMARY` parameter:
    
    ```
    START HADR ON DB dbname AS PRIMARY
    ```

11. To verify that HADR is running, you can query the status of the databases from the primary on NODE1 by running the `db2pd` command, as shown below, which returns all information about the HADR setup, including the standbys.
    
    ```
    db2pd -db dbname -hadr
    ```

    The standby starts in the local catchup mode replaying the local logs first. Then it enters the remote catchup pending state and goes to the remote catchup state once the primary starts and sends the logs to the standby. After all the logs on disk on the primary are replayed on standby, if the standby is in SUPERASYNC mode then HADR remains in remote catchup. If the principal standby is in any mode other than SUPERASYNC, then it enters the peer state. Read clients can connect to the standby server as long as the standby is not in the local catchup mode or replay only mode.

**Managing HADR**

Before starting HADR, make sure the respective DB2 instances are started. The instances can be started in any order. When starting HADR, though, start the standby first before the primary. The primary HADR start-up, without the `BY FORCE` option, requires the standby to be active within the `HADR_TIMEOUT` period. Otherwise, the start-up process fails to prevent a split-brain scenario.
The start command

The syntax for the start command is:

```
START HADR ON DATABASE database-alias [USER username [USING password]] AS {PRIMARY [BY FORCE] | STANDBY}
```

The BY FORCE option on the primary HADR start-up specifies that the HADR primary database need not wait for the standby database to connect to it. After you run start BY FORCE, the primary database still accepts valid connections from the standby database whenever the standby later becomes available.

For example, to start HADR on the primary, run the following command:

```
db2 start hadr on database &lt;dbname&gt; as primary
```

It is important to note which database you are on when you start HADR.

Although you can use the STOP HADR command to stop HADR on the primary or the standby, or both, it should be used with caution. If you want to stop the specified database but still want it to maintain its role as either an HADR primary or a standby database, do not run STOP HADR. If you run STOP HADR, the database becomes a standard database and might require reinitialization to resume operations as an HADR database. Instead, run the DEACTIVATE DATABASE command.

You can stop HADR from the command line or IBM Data Studio. When you want to bring your database to a standard database, run STOP HADR, which has the following syntax:

```
STOP HADR ON DATABASE database-alias [USER username [USING password]]
```

For example:

```
db2 stop hadr on database &lt;dbname&gt;
```

If a STOP is issued on an inactive primary database, the database becomes a standard database. If it was issued on an inactive standby database, the standby becomes a standard database and is placed in a roll forward pending state.

Takeover operations

You can perform a planned takeover, where you can switch roles, or you can takeover by force. Planned takeover is commonly done during an upgrade. Remember to reroute your clients either manually or by using ACR after you run TAKEOVER. Switching roles is done only from the standby when the databases are in the Peer state. The TAKEOVER command fails if the databases are in any other state.

For example, the following command is issued on a standby database to switch roles.

```
db2 takeover hadr on database &lt;dbname&gt;
```
A *takeover by force* is also called *failover*, which is issued from the standby database with the `TAKEOVER` command including the `BY FORCE` option. This operation is performed if the primary is not functional. The `BY FORCE` option tells the standby to become the new primary without contacting the original primary, as it does with a planned takeover. When the `PEER WINDOW ONLY` sub-option is specified, there is no committed transaction loss if the command succeeds and the primary database is stopped before the end of the peer window period.

To implement failover:

1. Ensure that the primary is down to minimize the chances of data loss. If a takeover by force is run and the primary is not down, this action can result in both the databases becoming primary databases. This situation is called *split-brain*.
2. Run `TAKEOVER HADR` with the `BY FORCE` and `PEER WINDOW ONLY` options on the standby database. The following is an example of this command for the SAMPLE database:

   ```
   db2 takeover hadr on database sample by force
   ```

In the case of a planned takeover, the HADR roles can be reversed by simply running the `takeover` command on the new standby. In the case of a forced takeover, often HADR might need to be reestablished on the old primary once it comes back up.

**DB2 pureScale feature**

The DB2 pureScale feature is designed to provide continuous HA to your critical distributed database environment. It also provides extreme capacity and application transparency. Multiple database servers, known as *members*, exist in a cluster and access a common shared disk system to process the incoming requests. This kind of capacity allows you to meet the most demanding business needs. The DB2 pureScale feature is built on the proven design features of DB2 z/OS sysplex architecture.

Figure 4 shows a typical pureScale setup.
Figure 4. pureScale setup

DB2 pureScale is a clustered database solution. It consists of a caching facility with primary and secondary servers, a set of DB2 servers (members), and a shared storage system (GPFS). All of these components in a cluster are supported by the IBM Tivoli SA MP. The cluster service runs on each member and monitors several resources, including:

- Access to paths and file systems
- Cluster caching facility server processes
- DB2 processes
- Host computers in the cluster
- Network adapters

This system is designed for fault tolerance, minimizes any component failures, and keeps your instance available. The cluster service provides a robust heartbeat detection system that ensures failed components are isolated rapidly. If a member failed while processing a request, that member is fenced off immediately. The data on the shared disk system remains available to other members for processing. Only in-flight data from the failed member is held until the member undergoes automated crash recovery and comes back up again. The automatic workload manager component of the pureScale feature distributes the workload to the rest of the members until the failed member is available again.
If the primary server in the caching facility fails, the secondary server takes over immediately until the primary recovers. After the primary is available, the failback is initiated to the primary. All of this is transparent to the application.

Any planned maintenance can also be done with no disruption to the instance availability. To perform maintenance on a member, quiesce the member, wait until transactions are complete, and take it offline. During this period, any requests coming in to the member will be redirected to other members. After the member is brought up online, it joins the other members and starts processing transactions.

**Backup and restore operations in a DB2 pureScale environment**

Running a `backup database` command in any member of the DB2 pureScale setup initiates a backup on all the members. A restore works the same way. A backup on the DB2 pureScale environment produces one image for the entire group because there is only one database partition. This image contains data from the table spaces and all the meta data and configuration information pertaining to all the members. If you are running an offline backup, then all members must be in a consistent state. Only one offline backup operation can be done at a time because the utility acquires super exclusive locks on all members. With online backup, concurrent backups are permitted but different backup operations cannot copy the same table spaces simultaneously.

The log archive path for the database must be a shared path with all the members so that during restore any member can retrieve the logs needed for roll forward.

The behavior of the backup and restore operations on a pureScale instance are described in Table 3.

### Table 3. Backup and restore operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Offline and consistent members</th>
<th>Offline and inconsistent members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online backup</td>
<td>The backup operation succeeds. The other member cannot become active while the backup utility is accessing the log file header (LFH) near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.</td>
<td>The backup operation succeeds, but it must wait for member crash recovery to be completed and for the other member to become either active or consistent. The other member cannot become active while the backup utility is accessing the LFH near the beginning of the backup operation or while the backup utility is accessing the log stream near the end of the backup operation.</td>
</tr>
<tr>
<td>Restore</td>
<td>The restore operation is completed normally.</td>
<td>The restore operation is completed normally.</td>
</tr>
</tbody>
</table>

Online backup with the `INCLUDE LOGS` option, which is the default, includes the set of log files needed to restore the backup image to the minimum recovery time. When you restore the image to a new database, the roll forward to end of logs often produces an error saying the next log file is missing. In most cases, this is OK. If you issue `ROLL FORWARD DATABASE` and `STOP` it will bring the database to a consistent state. However, if the command fails to stop the roll forward, then it means that the logs needed to restore the database to its minimum recovery time are missing.
Disaster recovery and high availability in a pureScale environment

Log shipping in a DB2 environment is the process of copying the transaction logs from one server to another standby site. You can have a clone of the primary database on the standby site keep rolling forward through the logs sent from the primary site, or just keep the logs and backup image on the standby site to recover in case of a disaster.

In a pureScale environment, each member maintains its own set of log files, called logstream, in a shared disk with separate log paths. When multiple applications access a member, they can generate dependent transactions. To make sense of these dependencies, DB2 must sort out all the transactions according to the time of updates. This ordering is called a log stream merge operation, which is required for crash recovery, database roll forward, and table space roll forward operations.

During a log stream merge operation, if DB2 determines there is a missing log file in the log streams then an error is returned. The roll forward utility returns SQL1273N. If a standby database is being kept up to date by applying logs as they are archived, the error can occur due to missing logs. This is common in the DB2 pureScale environment with log shipping. After the required logs are archived, the roll forward will complete successfully. In case of a disaster, some log files that are needed on the standby might not have been archived on the primary, or the logs might have been lost for some unexpected reason.

It is important to validate the end point of a roll forward by issuing the ROLLFORWARD...QUERY command. If the end point is acceptable, issuing the ROLLFORWARD...STOP will complete the roll forward. A restore of a DB2 pureScale instance backup on a non pureScale environment is not supported.

The following restrictions apply to the pureScale environments:

- A database that resides outside of a DB2 pureScale environment can be migrated to a DB2 pureScale environment. You cannot use database restore operations to migrate such a database to a DB2 pureScale environment.
- Snapshot backup operations using DB2 Advanced Copy Services (ACS) are not supported.

Examples

To back up a database called testdb with 1 member, use BACKUP DB TESTDB

To back up a database called testdb with 4 members, use BACKUP DB TESTDB

To restore the backup image whether it was from 1 member or multiple members, use RESTORE DB TESTDB

Or, issue the recover database command from any member with RECOVER DB TESTDB TO END OF LOGS

If the database does not exist, then use the RESTORE and ROLLFORWARD commands on any member to restore from the backup.
Conclusion

High availability entails the ability of the database server to handle unplanned and planned outages, with minimum impact to users, and the ease of recovery from system failures to maintain business continuity. Designing a highly available database solution requires an understanding of the applications accessing the database. It also involves a business decision after considering costs and the degree of availability required. This tutorial explored the various HA and recovery mechanisms available with the DB2 products such as the DB2 crash recovery, version recovery, roll-forward recovery, database and table space level backup operations, the DB2 HADR feature, and the DB2 pureScale feature.

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)