

DB2 9 DBA exam 731 prep, Part 7: High availability: Split mirroring and HADR

Skill Level: Intermediate

[Michael Dang \(dangm@ca.ibm.com\)](mailto:dangm@ca.ibm.com)

DB2 Database Administrator
IBM

[Sylvia Qi \(sylviaqi@ca.ibm.com\)](mailto:sylviaqi@ca.ibm.com)

WebSphere Application Server Function Verification Tester
IBM Toronto Lab

19 Jul 2006

This is the last in a [series of seven tutorials](#) to help you prepare for the DB2® 9 for Linux®, UNIX®, and Windows™ Database Administration (Exam 731). This tutorial focuses on two advanced high availability topics: split mirroring and high availability disaster recovery (HADR). With split mirroring you can restore databases using disk copies. HADR provides failover support, similar to the failover capability provided by HACMP and Microsoft™ Cluster Server.

Section 1. Before you start

About this series

If you are preparing to take the DBA certification exam 731, you've come to the right place -- a study hall, of sorts. This [series of seven DB2 certification preparation tutorials](#) covers the major concepts you'll need to know for the test. Do your homework here and ease the stress on test day.

About this tutorial

This tutorial focuses on two advanced high availability topics: Split mirroring and high availability disaster recovery (HADR). Combined with the sixth tutorial, High availability -- backup and recovery, it covers the objectives in section six of the exam, entitled "High Availability". You can view these objectives at:

<http://www-03.ibm.com/certify/tests/obj731.shtml>.

Objectives

In this tutorial, learn:

- The concept of high availability
- How log shipping works
- How split mirroring works and how to use a split mirror to provide high availability
- How HADR works and how to setup an HADR system
- How to update your system online using dynamic configuration parameters

Prerequisites

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

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

You should also be familiar with the following terms:

- *Object*: Anything in a database that can be created or manipulated with SQL (for example, tables, views, indexes, packages).
- *Table*: A logical structure that is used to present data as a collection of unordered rows with a fixed number of columns. Each column contains a set of values, each value of the same data type (or a subtype of the column's data type); the definitions of the columns make up the table structure, and the rows contain the actual table data.
- *Record*: The storage representation of a row in a table.
- *Field*: The storage representation of a column in a table.
- *Value*: A specific data item that can be found at each intersection of a row

and column in a database table.

- *Structured Query Language (SQL)*: A standardized language used to define objects and manipulate data in a relational database. (For more on SQL, see the fourth tutorial in this series.)
- *DB2 optimizer*: A component of the SQL precompiler that chooses an access plan for a Data Manipulation Language (DML) SQL statement by modeling the execution cost of several alternative access plans and choosing the one with the minimal estimated cost.

To take the DB2 9 DBA exam, you must have already passed the [DB2 9 Fundamentals exam 730](#). We recommend taking the [DB2 Fundamentals tutorial series](#) before starting this series.

System requirements

You do not need a copy of DB2 to complete this tutorial. However, you will get more out of the tutorial if you download the free trial version of [IBM DB2 9](#) to work along with this tutorial.

Section 2. The concept of high availability

High availability (HA) is the term used to describe systems that are operational and are available to users almost all the time. An HA system has the following characteristics:

- Efficient transaction processing, without performance degradation during peak operating periods.
- Quick recovery when hardware or software failures occur, or even when disasters strike -- if a proven backup and recovery strategy is in place.
- Failover capability. If the current database manager goes down, there must be another database manager to pick up the workload and make the database available for use almost instantaneously. Minimal interruption to client applications must be ensured.

While the previous tutorial covered the basic backup recovery strategies, in this tutorial, the focus is on the third characteristic of a high availability system -- failover capability. You learn about log shipping, split mirroring, and high availability disaster recovery (HADR).

Section 3. Log shipping

Primary and standby databases

An HA system usually consists of a primary database and a standby database. The database that is running currently is referred to as the *primary database*. All transactions go through this database. The *standby database* is a mirror of the primary database. If the primary database fails, the standby database takes over the existing transactions and becomes the new primary database.

What is log shipping?

Log shipping is a method where transaction logs are automatically backed up from a primary DB2 server and made accessible to a standby server.

The standby database is usually initialized by restoring a backup image of the primary database. However, the primary database continues to process transactions after the standby database has been initialized. If these transactions are not applied to the standby database, the standby database is not identical to the primary database. To keep the standby database in sync with the primary database, the standby database must have the ability to continuously apply the logs produced by the primary database. When it is time to takeover the primary database, the standby database can do so right away, without having to apply all the logs at the time of the takeover.

The `ROLLFORWARD DATABASE` command is used to apply log files produced by the primary database to the standby database. In fact, the command must be invoked continually on the standby database, so the logs are applied when they become available. This process can be scripted and scheduled to run periodically. Use archival logging in an HA system. Circular logging cannot be used because it does not provide the rollforward capability.

The question is: How can you make the log files produced by the primary database available to the standby database so you can apply them continuously? The answer is: log shipping.

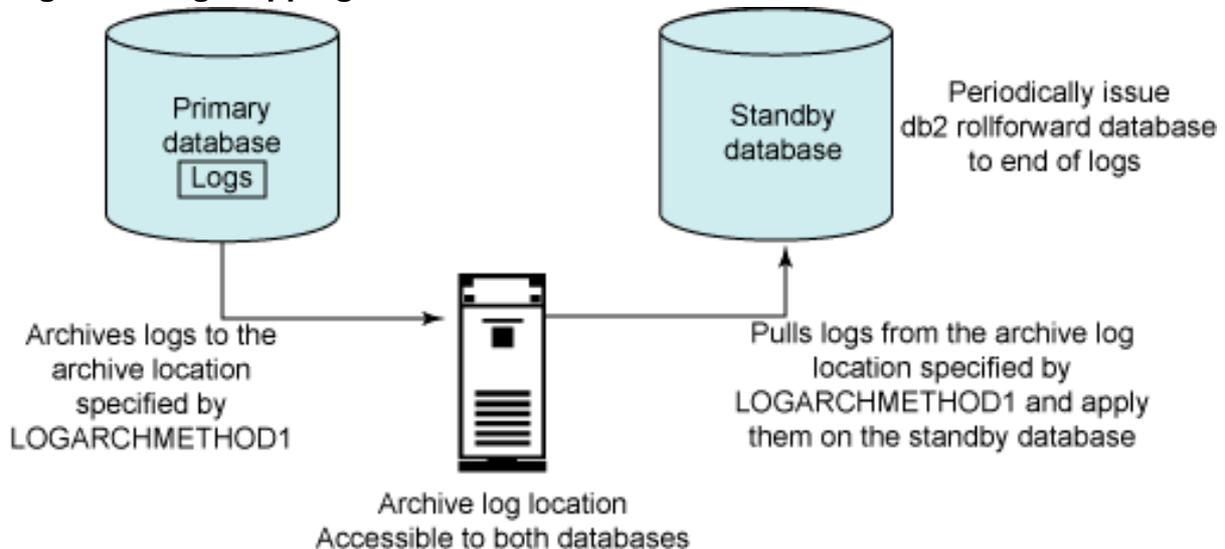
Setting up log shipping

The key to setting up log shipping is to configure the primary database so that it archives logs to a location that is accessible by the standby database.

Establish the log archival location on the primary database by setting the `LOGARCHMETH1` database parameter. Select a location that is accessible from the standby server, such as a mounted network drive or shared drive.

On the standby database, set the `LOGARCHMETH1` parameter to the same value as that on the primary database. When a `ROLLFORWARD DATABASE` command is issued on the standby database, DB2 pulls the logs from this archival location and applies them to the standby database.

Figure 1. Log shipping



To ensure redundancy, configure the `LOGARCHMETH2` parameter on the primary database. When `LOGARCHMETH2` is set, logs are archived to both locations set by `LOGARCHMETH1` and `LOGARCHMETH2`.

For log shipping to work, both systems must be running the same version of DB2.

Another way to set up log shipping is to use a `userexit` program on the standby database to continuously retrieve the archived log files from the primary database. If a `userexit` is used, both the primary and the standby databases must be configured with the same `userexit` program. The `userexit` is an old feature and its functionality has been replaced by `LOGARCHMETH1` since version 8.2. However, for backwards compatibility reasons, the `userexit` program method is still supported in version 8.2 and version 9.

Section 4. High availability through online split mirroring and suspended I/O Support

Split mirror and suspended I/O

In the backup and recovery tutorial, you learned that an online backup lets users maintain their connections to the database while the database is being backed up.

Although this meets the criterion of a high availability system, this process can be time consuming for large databases. There is another backup recovery strategy that is faster than online backups -- split mirroring.

With *split mirroring*, instead of taking a backup of the database using the DB2 backup utility, you make a disk copy of the database directories and then restore this disk copy when needed. This process has the following advantages over the traditional database backup recovery method:

- It eliminates backup operation overhead from the primary machine, which could be time consuming if the database is large.
- The restore process using a split mirror is faster than using the restore utility.

Split mirror

Splitting a mirror means creating an "instantaneous" copy of the source database by writing the data to a hard disk. When required, this disk copy can be used to clone a new, but identical, database or used as a backup copy to restore the original database.

The method you choose to split the mirror is not within the control of DB2. Take a file system copy of the database directory if you wish. It is also recommended to use any intelligent storage devices such as the IBM Enterprise Storage Server (ESS), known as Shark and EMC Symmetrix 3330. Using the FlashCopy technology, the ESS can establish near-instantaneous copies of the data entirely within itself. The instant split feature of EMC TimeFinder software on Symmetrix is also capable of splitting a mirror copy in a similar manner.

A split mirror of a database includes the entire contents of the database directory, all the tablespace containers, and the local database directory. The active log directory might be included, depending on how you want to use this split mirror image. Read more on this topic later.

Suspended I/O

When splitting a mirror, it is important to ensure that there are no page writes occurring on the database. DB2 suspended I/O support lets you perform split mirror operations without having to shut down the database. The idea is to put the database in a write suspend mode before splitting the mirror, and after the split, resume normal I/O activities.

While the database is in write suspended mode, all its table spaces are placed in SUSPEND_WRITE state. All operations continue to function normally. However, some transactions may have to wait if they require disk I/O. These transactions proceed normally once the write operations on the database are resumed.

The following commands are used to suspend and resume write operations on a database:

To suspend write operations, issue:

```
CONNECT TO database-alias
SET WRITE SUSPEND FOR DATABASE
```

To resume write operations, issue:

```
SET WRITE RESUME FOR DATABASE
```

The db2inidb tool

Now you have a mirror of the source database, which is basically a disk copy. You cannot use the `RESTORE DATABASE` command on this disk copy to restore any databases because this is not a DB2 database backup. It is merely a disk copy of the database files. To initialize the disk copy into a usable DB2 database, use the `db2inidb` command:

```
DB2INIDB database-alias
AS {SNAPSHOT | STANDBY | MIRROR}
[RELOCATE USING config_file]
```

You can initialize a mirror in three different ways:

- Snapshot: Creates a clone of the source database
- Standby: Creates a standby database
- Mirror: Restores the original source database

Both the Snapshot and Standby options create a new, but identical, database to the source database using the mirror image. Therefore, the split mirror database cannot exist on the same system as the source database because it has the same structure and uses the same instance name as the source database. If the split mirror database must exist on the same system as the source database, specify the `RELOCATE USING configuration-file` option when issuing the `db2inidb` command.

The format of the relocate configuration file (text file) follows. Use the configuration file to relocate the database directory structures:

```
DB_NAME=oldName,newName
DB_PATH=oldPath,newPath
INSTANCE=oldInst,newInst
NODENUM=nodeNumber
LOG_DIR=oldDirPath,newDirPath
CONT_PATH=oldContPath1,newContPath1
CONT_PATH=oldContPath2,newContPath2
```

Cloning a database using the db2inidb snapshot option

This option creates an instantaneous copy of the source database at the time when I/O is suspended. Hence, the name *snapshot*. During the initialization process, the split mirror database goes through a crash recovery. After the crash recovery is completed, the database is ready for use right away. Any outstanding uncommitted work at the time of the split mirror is rolled back.

The steps to create a clone database using the db2inidb snapshot option are:

1. Suspend I/O on the source database:

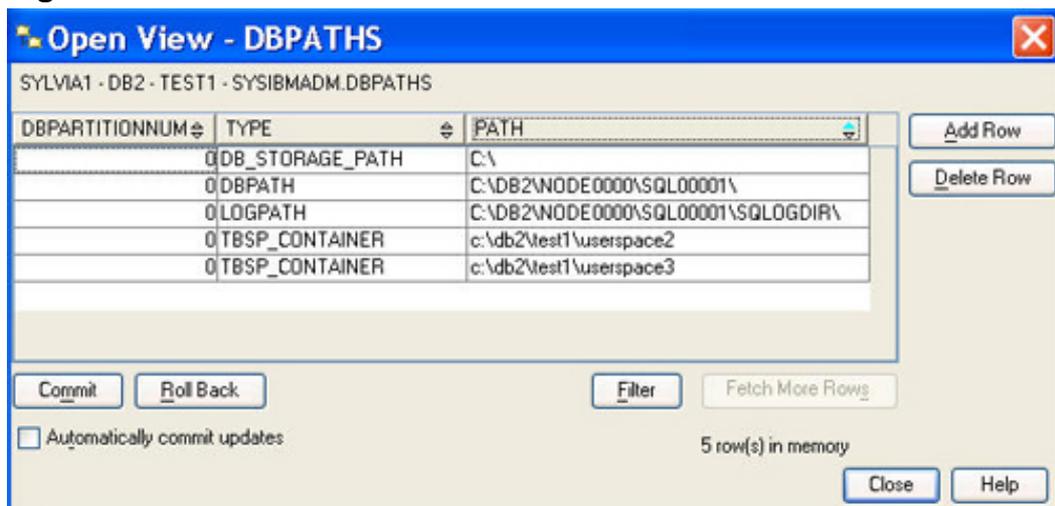
```
CONNECT TO source-database-alias
SET WRITE SUSPEND FOR DATABASE
```

2. Split the mirror. To split the mirror, use the file system copy method or any vendor products mentioned earlier. If you choose to use a vendor product, make sure you consult the documentation applicable to your device on how to create a split mirror. Regardless of the variations on the split mirror process, all of the following must be split mirrored at the same time:

- The entire contents of the database directory
- All the table space containers
- The local database directory
- The active log directory, if it does not reside in the database directory

All of the preceding information is stored in the system view SYSADM.DBPATHS. Run a `select * from sysadm.dbpaths` statement to obtain this information. You can also view the contents of this view from the Control Center. The following figure shows a sample content of the view:

Figure 2. The SYSADM.DBPATHS view



The SYSADM.DBPATHS view provides the following information:

- The database directory is C:\DB2\NODE0000\SQL00001.
- The tablespace container paths are c:\db2\test1\userspace3 and c:\db2\test1\userspace2.
- The local database directory is C:\DB2\NODE0000\SQLDBDIR (This is not shown in the figure, but the local database directory is the SQLDBDIR directory at the same level as the database directory.
- The active log directory is C:\DB2\NODE0000\SQL00001\SQLOGDIR.

All these directories must be split mirrored.

3. Resume I/O on the source database. Issue the following command to resume I/O activities on the source database. Use the same connection session from step 1 when issuing this command:

```
SET WRITE RESUME FOR DATABASE
```

4. Make the split mirror accessible:
 - a. On the target machine, create the same DB2 instance as it is on the source machine.
 - b. Restore split mirror obtained from step 2 to exactly the same paths as they are on the source machine. If the split mirror is on a network drive, mount the network drive to the target machine.
 - c. Run the following command to catalog the database on the target machine:

```
CATALOG DATABASE database-name AS database-alias ON path
```

where:

database-alias must match the database alias of the source database

path must match the database path of the source database (Use the LIST DB DIRECTORY command to display the database path, or check the DB_STORAGE_PATH field in the SYSADM.DBPATHS view, as shown in Figure 2.)

5. Initialize the split mirror database to be a clone database
 - a. Start the instance on the target machine using the db2start command.

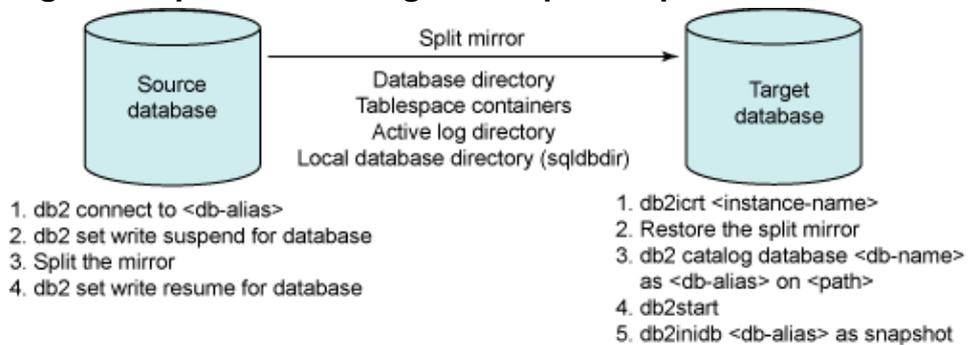
- b. Initialize the split mirror database using the snapshot option:

```
DB2INIDB database-alias AS SNAPSHOT
```

The preceding db2inidb command initiates a crash recovery, which rolls back all uncommitted transactions at the time of the split mirror, thereby making the database consistent. It is essential to have all the log files from the source that were active at the time of the split. The active log directory must not contain any log file that is not a part of the split mirror. After the completion of the crash recovery, the database is available for operation.

The Figure 3 summarizes the split mirror process using the snapshot option:

Figure 3. Split mirror using the snapshot option



Creating a standby database using the db2inidb standby option

The db2inidb standby option creates a standby database of the source database, also known as the primary database.

When a split mirror is initialized as a standby database, it is immediately placed in rollforward pending state. Continually apply the inactive log files from the primary database as they become available, keeping the standby database current with the primary database. The log shipping method discussed earlier is used here to make the logs available to the standby database. If a failure occurs on the primary database, use the standby database to take over the primary database role.

The steps to creating a standby database using the db2inidb standby option are:

1. Suspend I/O on the primary database. Follow the same steps as in the snapshot scenario to suspend I/O on the primary database.
2. Split the mirror. Use the appropriate method to split mirror the primary database. The split mirror must contain the following:
 - The entire contents of the database directory

- All the table space containers
- The local database directory

You do NOT need to split the active log directory in this scenario. How logs are handled is covered later.

3. Resume I/O on the source database. Follow the same steps as in the snapshot scenario to resume I/O on the primary database.
4. Make the split mirror accessible. Follow the same steps as in the snapshot scenario to make the split mirror accessible.
5. Initialize the split mirror database as a standby database. The following command initializes the database and puts it in a rollforward pending state, so logs from the primary database can be applied.

```
DB2INIDB database-alias AS STANDBY
```

6. Continuously apply the logs that have been archived by the primary database to the standby database using the `ROLLFORWARD DATABASE` command. To keep the standby database as current as possible, new inactive log files (these are logs that have been archived) from the primary database should be continually applied on the standby database as they become available. Do this by issuing the `ROLLFORWARD DATABASE` command on the standby database without using the `STOP` or `COMPLETE` option. To make the logs files accessible to the standby database, use the log shipping method discussed in [Setting up log shipping](#).

Continuously apply the archived logs to the standby database using the `ROLLFORWARD DATABASE` command:

```
ROLLFORWARD DB database-alias TO END OF LOGS
```

7. Bringing the standby database online.
In case of a failure on the primary database, you want the standby database to take over the primary database role by switching the database online. To switch the standby database online, perform the following:
 - a. Make the active log path accessible to the standby database. In step 6, you have only retrieved and applied the archived logs produced by the primary database to the standby database. You have not applied the active logs. When you are ready to bring the

standby database online, retrieve the active logs from the primary database and apply them on the standby. This retrieval process can be done manually, that is, copy the active logs from the primary database to the logpath directory of the standby server.

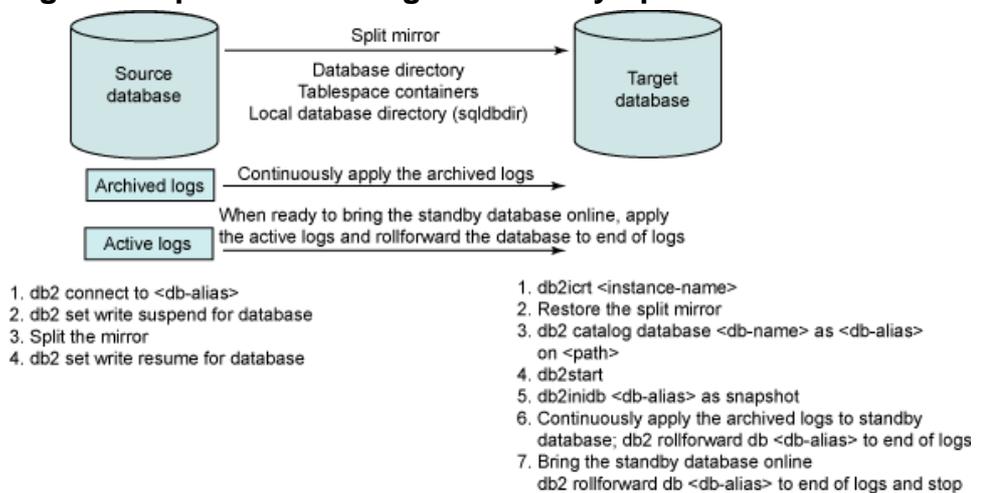
- b. Rollforward database to end of logs and stop:

```
ROLLFORWARD DB database-alias TO END OF LOGS AND STOP
```

After the roll forward process is completed, the database is ready for use.

The following figure summarizes the split mirror process using the standby options:

Figure 4. Split mirror using the standby option



Creating a backup image of the source database using the db2inidb mirror option

The mirror option of the db2inidb command is used to create a quick mirror file backup of the source database. The split mirror can be used to restore the source database if needed. This procedure can be used instead of performing the backup and restore database operations on the source database.

The steps to create a backup image of the source database using the db2inidb mirror option are:

1. Suspend I/O on the source database. Follow the same steps as in the snapshot scenario to suspend I/O on the source database.
2. Split the mirror. Use the appropriate method to split mirror the source database. The split mirror must contain the following:

- The entire contents of the database directory
- All the table space containers
- The local database directory

You do NOT need to split the active log directory in this scenario.

3. Resume I/O on the source database. Follow the same steps as in the snapshot scenario to resume I/O on the source database.
4. Restore the source database using the split mirror image. There is no "target" database in this scenario. The intent of the mirror option is to use the mirror copy to recover the source database when needed.
 - a. Stop the instance using the db2stop command.
 - b. Copy the data files of the split mirror database over the original database.
 - c. Start the instance using the db2start command.
 - d. Issue the following command to initialize the split mirror database. This replaces the source database with the split mirror image and places it into a rollforward pending state, so logs can be re-applied.

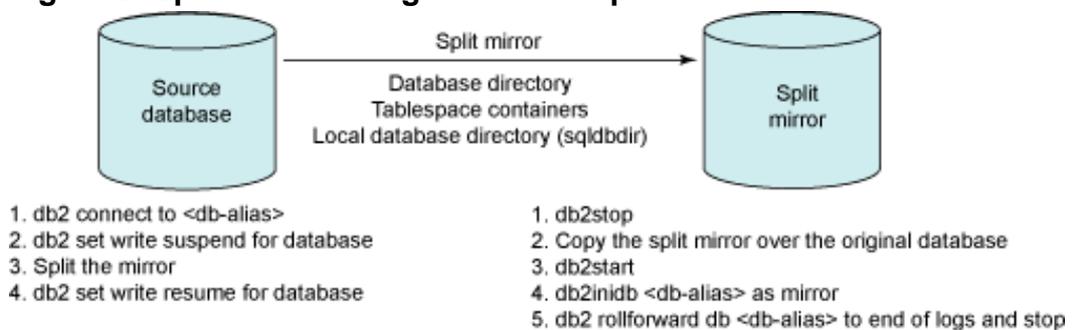
```
DB2INIDB database-alias AS MIRROR
```

- e. Roll forward the database to end of logs. After the roll forward process is completed, the database is ready for use.

```
ROLLFORWARD DB database-alias TO END OF LOGS AND STOP
```

Figure 5 summarizes a split mirror procedure using the mirror option:

Figure 5. Split mirror using the mirror option



Split mirroring in a partitioned environment

In a partitioned database environment, suspend the I/O on each partition during the split mirror process. Resume the I/O on each partition afterwards. The same applies to the db2inidb tool, which must be run on each mirrored partition to initialize the database.

Because each partition is treated independently, the partitions can be suspended independently of one another. That means you do not need to issue a `db2_all` to suspend I/O on all of the partitions. If each partition is to be suspended independently, suspend the catalog partition last. This is because an attempt to suspend I/O on any of the non-catalog nodes requires a connection to the catalog partition for authorization. If the catalog partition is suspended, then the connection attempt may hang.

The db2inidb tool does not require any connections to the database. Therefore, you can run the tool independently on each split mirror or use `db2_all` to run it on all partitions simultaneously. The only requirement is for the database manager to be started.

To suspend writes on a database with 3 partitions, 0, 1, 2, with partition 0 being the catalog partition, issue the following command:

```
export DB2NODE=1
db2 terminate
db2 connect to database-alias
db2 set write suspend for database

export DB2NODE=2
db2 terminate
db2 connect to database-alias
db2 set write suspend for database

export DB2NODE=0
db2 terminate
db2 connect to database-alias
db2 set write suspend for database
```

To run db2inidb on all partitions simultaneously, issue:

```
db2_all "db2inidb database-alias as db2inidb-option"
```

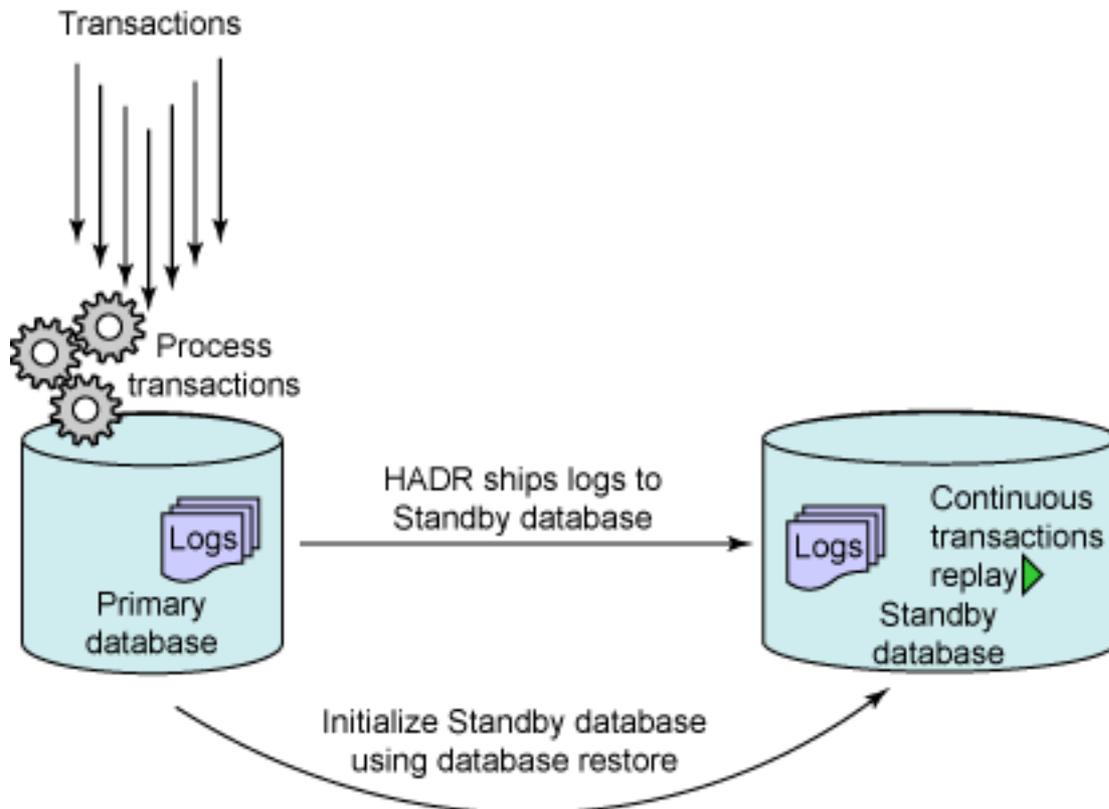
Section 5. High availability disaster recovery (HADR)

DB2 High availability disaster recovery (HADR) is a database replication feature that provides a high availability solution. HADR is currently available in the single-partitioned database environment only.

Two machines are involved in an HADR setup: the primary and secondary. The primary machine is where the source database is stored. As transactions are

processed at the source database, database logs are automatically shipped to the secondary server. The secondary server stores a database that is cloned from the source database. It can be initialized using database restore or by split mirroring. When HADR is started, log files are received and they are replayed on the secondary. Through continuous log replay, the secondary database keeps a replica of the primary database and acts as a standby database.

Figure 6. An overview of HADR



When a failure occurs on the primary, the standby database takes over the transactional workload and becomes the new primary database. If the failed machine becomes available again, it can be resynchronized and catch up with the new primary database. The old primary database then becomes the new standby database.

Figure 7. Standby database taking over primary role

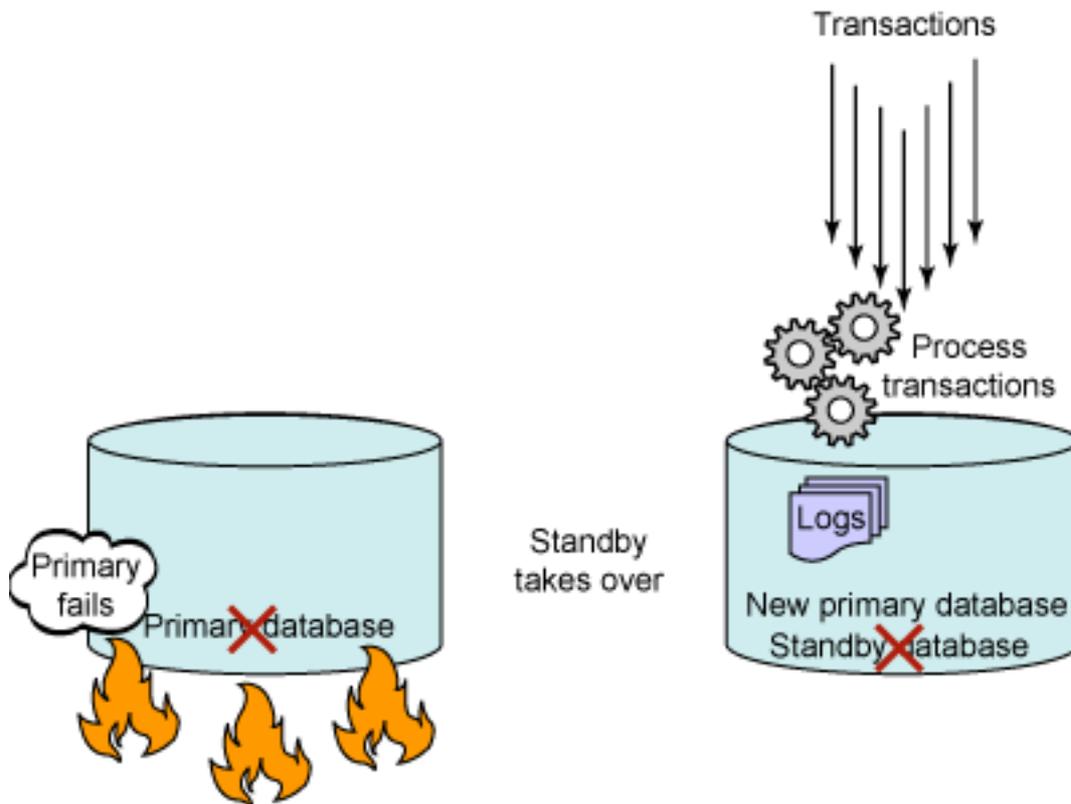
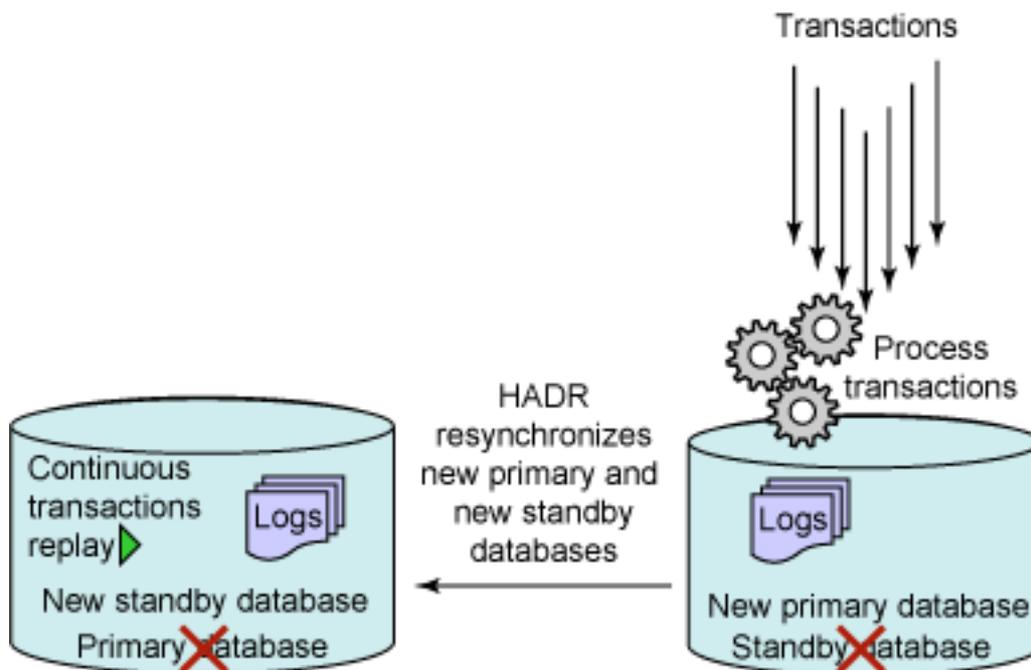


Figure 8. New standby database resynchronizes and catches up with the new primary DB



HADR database configuration parameters

To set up an HADR environment, update certain database configuration parameters.

First of all, archival logging must be enabled on the primary database. See the

[Backup and Recovery tutorial](#) to learn how to enable archival logging.

Table 1 lists descriptions of the database configuration parameters related to HADR. An example of how to update them appears later in the tutorial.

Table 1. HADR-related database configuration parameters

HADR-Related DB CFG	Description
HADR_LOCAL_HOST	Specifies the local host for HADR TCPIP communication. Either a host name or an IP address can be used.
HADR_LOCAL_SVC	Specifies the TCPIP service name or port number for which HADR process accepts connections on the local host. This port cannot be the same as the SVCENAME or SVCENAME +1 of the HADR instance.
HADR_REMOTE_HOST	Specifies the TCPIP host name or IP address of the remote HADR node.
HADR_REMOTE_SVC	Specifies the TCPIP service name or port number for which HADR process accepts connections on the remote node. This port cannot be the same as the SVCENAME or SVCENAME +1 of the remote instance.
HADR_REMOTE_INST	Specifies the instance name of the remote server. Administration tools, such as the DB2 Control Center, use this parameter to contact the remote server.
HADR_TIMEOUT	Specifies the time (in seconds) that the HADR process waits before considering a communication attempt to have failed. The default value is 120 seconds.
HADR_SYNCMODE	Specifies the synchronization mode. It determines how primary log writes are synchronized with the standby when the systems are in peer state. Valid values are: SYNC, NEARSYNC, ASYNC. The default value is NEARSYNC. (Peer state is discussed later

	in this section)
HADR_DB_ROLE	Specifies the current role of a database. Valid values are: STANDARD, PRIMARY, and STANDBY. STANDARD means the database is not HADR-enabled.

HADR database states

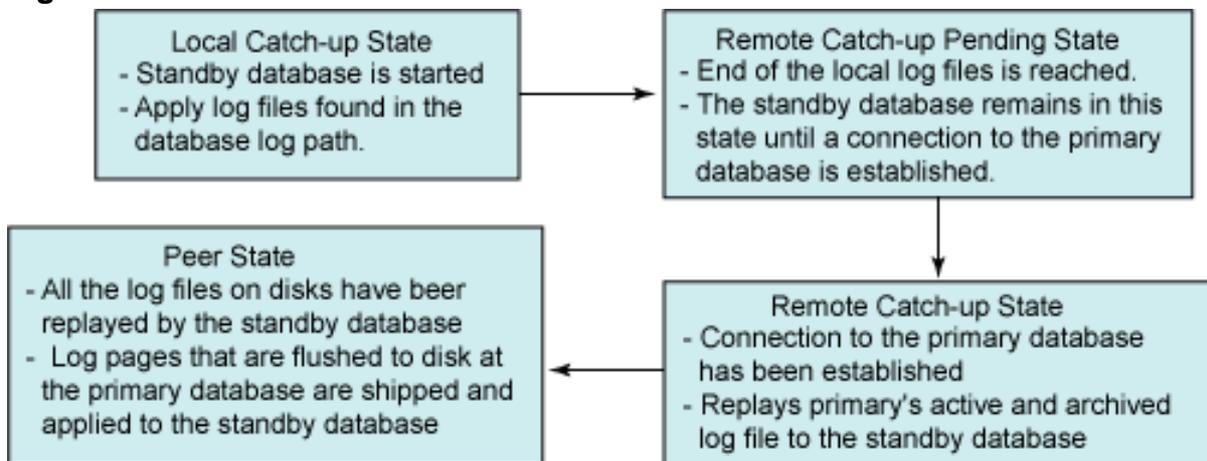
When the standby database is first started, it enters the local catch-up state. Log files (if any) found in the local log path are read and replayed to the standby database.

When the end of local log file is reached, the standby database enters the remote catch-up state. It replays log pages from the primary's archived logs as well as the active logs until the standby database is caught up to the last active log.

When all of the log files on the primary system have been replayed, the primary and standby databases enter the peer state.

In the peer state, log pages are shipped and applied to the standby database whenever the primary database flushes log pages to disk. Specify one of the three synchronization modes to protect from potential loss of data. Details of the synchronization modes are discussed in the next section.

Figure 9. HADR database states



HADR synchronization modes

Recall that when an HADR pair is in the peer state, log pages that are flushed to the log file on disk at the primary database are shipped and applied to the standby database. To indicate how log writing is managed between the primary and standby databases, a synchronization mode is specified. There are three synchronization modes: SYNC (Synchronous), NEARSYNC (Near Synchronous), and ASYNC

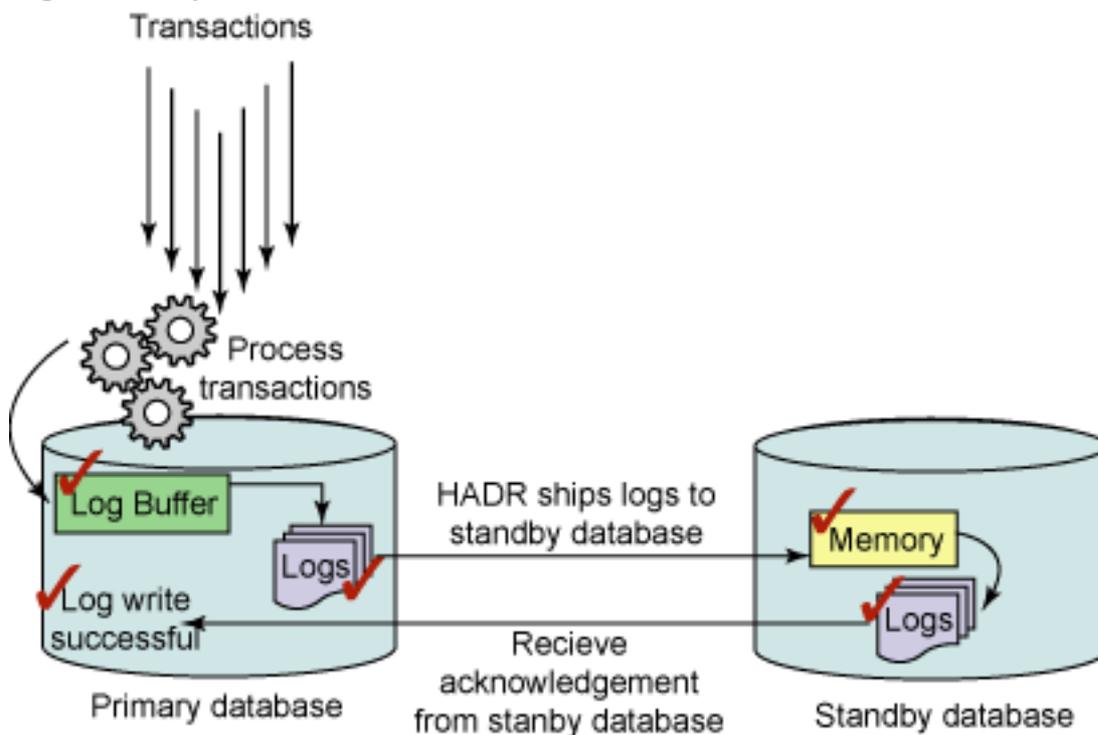
(Asynchronous).

SYNC

In synchronous mode, log writes are considered successful only when:

- Logs are written to log files on the primary database
- The primary database has received acknowledgement from the standby database that logs are successfully written to log files on the standby database

Figure 10. Synchronization mode -- SYNC

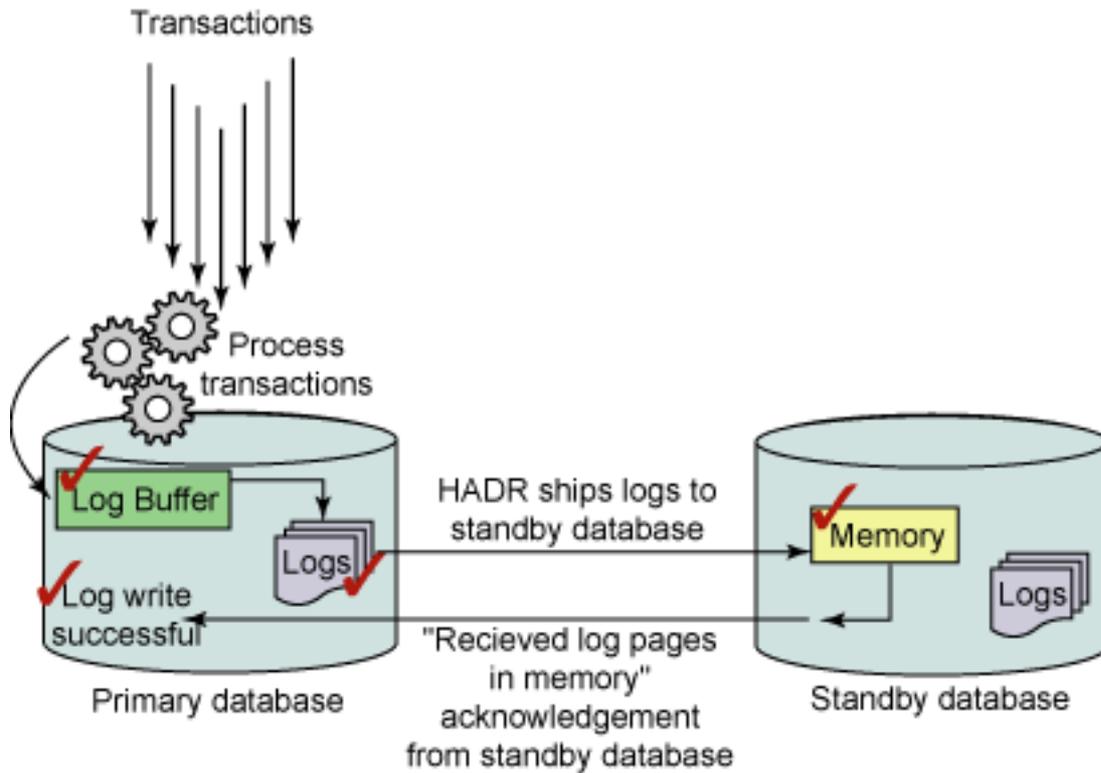


NEARSYNC

Log records in the primary and standby databases are almost (or near) synchronous because log writes are considered successful only when:

- Log records have been written to the log files on the primary database
- The primary database has received acknowledgement from the standby database that logs are successfully written to main memory on the standby database

Figure 11. Synchronization mode -- NEARSYNC

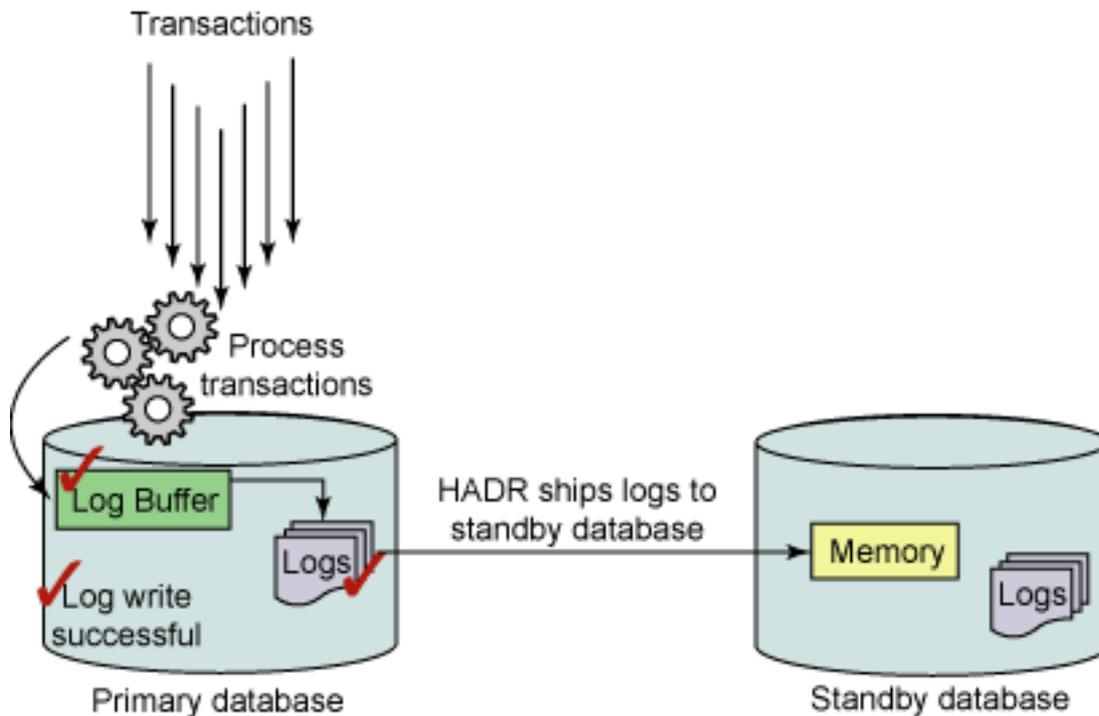


ASYNC

In this mode, the primary database does not wait for acknowledgement from the standby database. Log writes are considered successful only when:

- Log records have been written to the log files on the primary database
- Log records have been delivered to the standby database; no acknowledgement is expected

Figure 12. Synchronization mode -- ASYNC



HADR commands overview

HADR is managed by three simple commands:

- START HADR
- STOP HADR
- TAKEOVER HADR

The syntax of the commands follows:

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

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

TAKEOVER HADR ON DATABASE database-alias
[USER userid] [USING password]
[BY FORCE]
```

Issuing the `START HADR` command with either the `AS PRIMARY` or `AS STANDBY` option changes the database role to the one specified if the database is not already in that role. This command also activates the database, if it is not already activated.

The `STOP HADR` command changes an HADR database (either primary or standby) into a standard database. Any database configuration parameters related to HADR remain unchanged so that the database can easily be reactivated as an HADR database.

The `TAKEOVER HADR` command, which you can issue on the standby database only, changes the standby database to a primary database. When you do not specify the `BY FORCE` option, the primary and standby databases switch roles. When you specify the `BY FORCE` option, the standby database unilaterally (without changing the primary to standby) switches to become the primary database. In this case, the standby database attempts to stop transaction processing on the old primary database. However, there is no guarantee that transaction processing will stop. Use the `BY FORCE` option to force a takeover operation for failover conditions only. To whatever extent possible, ensure that the current primary has definitely failed or shut it down, prior to issuing the `TAKEOVER HADR` command with the `BY FORCE` option.

Initializing HADR

Now that you have learned the concept of HADR, you are ready to learn how to set up an HADR environment.

In Table 2, you are going to set up an HADR environment for a database called TEST1. TEST1 resides on database server `server1.torolab.ibm.com`, under instance `DB2INST1`. The service port number, or `SVCENAME`, of `DB2INST1` is 50000.

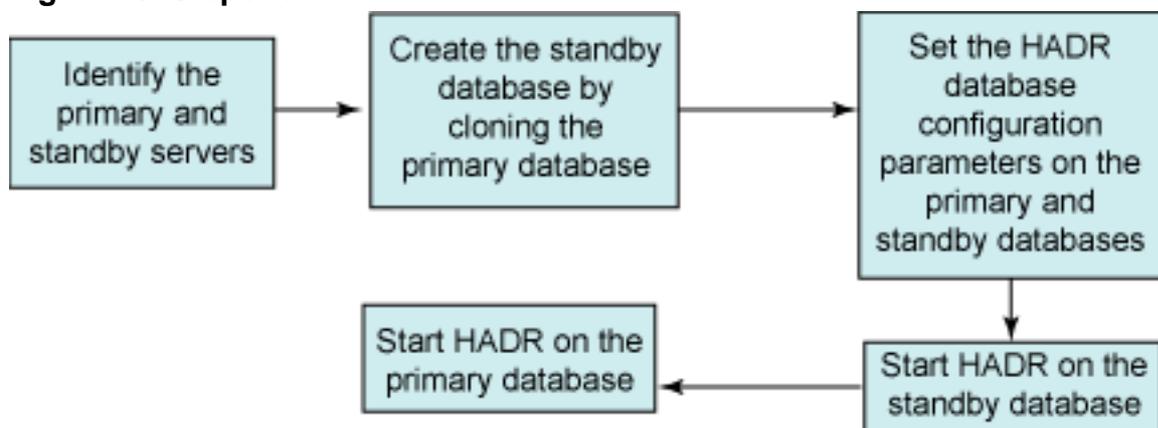
Use another database server, `server2.torolab.ibm.com`, as the standby server. TEST1 currently does not exist on the standby server.

Table 2. Setting up an HADR environment

Server Name	Instance Name	SVCENAME or Port Number	Database Name
<code>server1.torolab.ibm.com</code>	<code>DB2INST1</code>	50000	TEST1
<code>server2.torolab.ibm.com</code>	<code>DB2INST1</code>	50000	--

To initialize HADR for the first time, perform the following five steps as Figure 13 shows:

Figure 13. Steps to initialize an HADR environment



1. Identify the primary and the standby servers. Determine the host name,

host IP address, and the service name or port number for each of the HADR databases.

You have already identified the primary and standby servers. The primary server is server1. The standby server is server2.

Make sure that these two servers can communicate with each other through TCPIP. Try pinging each other using their hostnames; there should be no errors. If the hostnames do not work, try using their TCPIP addresses. Use the alternative that works best for your HADR configuration.

2. Create the standby database by cloning the primary database.

There are two options to clone the primary database:

- One option is to take a backup of the primary database and restore it on the standby server. This option uses the standard backup and restore method.
- Another option to clone the primary database is by split mirroring. Make a split mirror of the primary database and initialize it on the standby server. When initializing the mirror, the standby option must be used with the db2inidb command for the mirror and the snapshot options to work.

In this example, use the backup/restore method to create the standby database.

First take a backup of the primary database, TEST1, on server1:

```
BACKUP DB test1
```

Next, ftp the backup image over to the standby server and restore it.

Strict symmetry of table space and container configuration is required on the standby database. The name, path, and size of the containers must match the primary database. The names of the databases must be the same. If any of the configurations does not match, HADR might fail to replicate the data to the standby database. Therefore, before restoring the database, make sure the standby server has the same architectural setup as the primary server.

Restore the backup image to the same location as it is on the primary server. On the primary server, a LIST DB DIRECTORY shows:

```
Database alias           = TEST1
Database name           = TEST1
Local database directory = C:
Database release level  = b.00
Comment                 =
Directory entry type    = Indirect
```

```
Catalog database partition number = 0
Alternate server hostname          =
Alternate server port number      =
```

The database resides on the C: drive on the primary server. When restoring the backup image on the standby server, restore it to C: drive as well. On the standby server, server2, issue the following command to restore the TEST1 database:

```
RESTORE DB test1 ON C:
```

After the restore is completed, the database is put in rollforward pending state. This is because the backup was taken from a database that is enabled for archival logging. DO NOT issue the `ROLLFORWARD` command to bring the database out of the rollforward pending state. The database must remain in this state in order to start HADR as a standby database properly.

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

- HADR_LOCAL_HOST
- HADR_LOCAL_SVC
- HADR_REMOTE_HOST
- HADR_REMOTE_SVC
- HADR_REMOTE_INST
- HADR_SYNCMODE
- LOGINDEXBUILD

Set the database configuration parameter `LOGINDEXBUILD` to `ON` in an HADR environment. Otherwise, any index creation, recreate, or reorganization on the current or future primary database server might not be recovered on the current or future standby database server using HADR. Those indexes which cannot be recovered are marked as invalid and are rebuilt implicitly either at the end of the HADR takeover process or after the HADR takeover process when the underlying tables are to be accessed.

On each server, assign an unused TCPIP port for HADR to use. There are no strict rules for choosing a port number for HADR; as long as it is not being used by another application and it is not equal to `SVCENAME` or `SVCENAME+1`, it can be used.

On the primary server, use port 70000 for HADR. Update the `Windows\System32\drivers\etc\services` file to add the following line:

```
Hadr_port          70000/tcp
```

On the standby server, use port 80000 for HADR. (You could use port 70000 if it meets the rules described in Table 1. For illustration purpose, we chose 80000 so it is easier to distinguish between the primary server port and standby server port.)

Add the following line in the services file on the standby server:

```
Hadr_port          80000/tcp
```

Use the default value for HADR_SYNCMODE, which is NEARSYNC.

Now you have all the information necessary to configure the HADR parameters. Issue the following commands to configure the parameters on the standby server:

```
UPDATE DB CFG FOR test1 USING HADR_LOCAL_HOST server2.torolab.ibm.com
UPDATE DB CFG FOR test1 USING HADR_LOCAL_SVC 80000
UPDATE DB CFG FOR test1 USING HADR_REMOTE_HOST server1.torolab.ibm.com
UPDATE DB CFG FOR test1 USING HADR_REMOTE_SVC 70000
UPDATE DB CFG FOR test1 USING HADR_REMOTE_INST db2inst1
UPDATE DB CFG FOR test1 USING LOGINDEXBUILD on
```

On the primary database server, configure the same set of parameters:

```
UPDATE DB CFG FOR test1 USING HADR_LOCAL_HOST server1.torolab.ibm.com
UPDATE DB CFG FOR test1 USING HADR_LOCAL_SVC 70000
UPDATE DB CFG FOR test1 USING HADR_REMOTE_HOST server2.torolab.ibm.com
UPDATE DB CFG FOR test1 USING HADR_REMOTE_SVC 80000
UPDATE DB CFG FOR test1 USING HADR_REMOTE_INST db2inst1
UPDATE DB CFG FOR test1 USING LOGINDEXBUILD on
```

4. Start HADR on the standby database using the START HADR command:

```
START HADR ON DB test1 AS STANDBY
```

Start the standby database before starting the primary database. If you start the primary database first, the startup procedure fails if the standby database is not started within the time period specified by the HADR_TIMEOUT database configuration parameter.

5. Start HADR on the primary database, as in the following example:

```
START HADR ON DB test1 AS PRIMARY
```

HADR is now started on the primary and standby databases.

When the primary server is first started, it waits for the standby server to contact it. If the standby server does not make contact with the primary after a specific period of time, HADR does not start. This timeout period is configurable using the `HADR_TIMEOUT` configuration parameter (see Table 1). The purpose is to avoid two systems accidentally starting up as primary at the same time.

During the HADR startup process, the databases go through the following states: local catch-up, remote catch-up pending, remote catch-up and peer. Logs are shipped automatically from the primary server to the standby server, and then replayed on the standby server. This is to ensure that the standby database is "up to date" with the primary database.

It is highly recommended that the database configuration parameters and database manager configuration parameters be identical on the systems where the primary and standby databases reside. Otherwise, the standby database does not perform the same way as the primary database.

One exception is the `LOGFILSIZ` database configuration parameter. The value of this parameter is ignored by the standby server. The standby server automatically creates log files that match the size of the log files on the primary database. This guarantees identical log files on both databases.

Performing a takeover -- switching database roles

Now you have set up an HADR environment. In the case where the primary server becomes unavailable, perform an HADR takeover operation and let the standby database take over the primary database role, ensuring minimal interruption to users that are connected. Perform an HADR takeover in one of the following two ways:

- Switch the database roles between the primary and standby, that is, standby becomes primary, primary becomes standby
- Failover from primary to standby

The database role switching option causes any applications currently connected to the HADR primary database to be forced off. This action is designed to work in coordination with the DB2 automatic client reroute feature. With automatic client reroute enabled, existing connections are forced off during the takeover and reestablished to the new primary server after the takeover. New client connections to the old primary database are automatically rerouted to the new primary server. (Automatic client rerouting is discussed in detail later.)

To switch database roles, issue the `TAKEOVER HADR` command on the standby database. The command can only be issued on the standby database and only when it is in peer state. Get the HADR status of a database using the `GET`

SNAPSHOT command. For example:

```
GET SNAPSHOT FOR DATABASE ON test1
```

The following command initiates an HADR database switching role operation on the standby database TEST1:

```
TAKEOVER HADR ON DB test1
```

Performing a takeover -- failover

The failover option does not switch database roles. During a failover operation, the standby database takes over the primary database role without having the primary database switch to standby.

Use this option only when the primary database becomes unavailable and you want the current standby database to become the new primary database. If the primary database is still active and you execute an HADR failover, you might end up with two primary databases or data might be lost.

Before issuing the `TAKEOVER` command, make sure the failed primary database is completely disabled. When a database encounters internal errors, normal shutdown commands might not completely shut it down. You might need to use operating system commands to remove resources such as processes, shared memory, or network connections.

If the primary database is not completely disabled, the standby database still sends a message to the primary database asking it to shutdown. The standby database then switches to the role of primary database whether or not it receives confirmation from the primary database that it has shutdown.

To perform an HADR failover, issue the `TAKEOVER HADR` command with the `BY FORCE` option on the standby database. Note that the command can only be issued on the standby database and only when it is in peer state or remote catch-up pending state.

The following command initiates an HADR failover operation on database TEST1:

```
TAKEOVER HADR ON DB test1 BY FORCE
```

Summary of the takeover behavior

Tables 3 and 4 summarize the behavior of the `TAKEOVER HADR` command when issued on the standby database, with respect to the HADR state it is in:

Table 3. Takeover behavior without BY FORCE option

Standby Database State	Takeover Behavior
Peer	The primary and standby databases switch roles
Local catch-up or remote catch-up	ERROR
Remote catch-up pending	ERROR

Table 4. Takeover behavior with BY FORCE option

Standby Database State	Takeover Behavior
Peer	The standby notifies the primary to shut itself (the primary) down. The standby stops receiving logs from the primary, finishes replaying the logs it has already received, and then becomes a primary. The standby does not wait for any acknowledgement from the primary to confirm that it has received the takeover notification or that it has shut down.
Local catch-up OR remote catch-up	ERROR
Remote catch-up pending	Standby db becomes a primary db

The automatic client reroute feature

By now, you have seen how to set up an HADR environment and how to perform a takeover using both the database role switching and the failover options.

You have learned how to make a database highly available with the HADR feature. If the primary database goes down, its standby database takes over and becomes the new primary.

However, this does not mean that the client applications are automatically aware of this takeover and are smart enough to connect to the new primary server instead of the old primary server following a takeover. In fact, with only HADR enabled, clients won't automatically connect to the new primary database following a takeover. They still try to connect to the old primary and fail because the primary sever has either

become the standby server (database role switching) or has been completed disabled (failover).

That is why the DB2 automatic client reroute feature is necessary. When an HADR environment is enabled with the automatic client reroute feature, after a takeover all current and new client connections are automatically rerouted to the new primary server so the applications can continue their work with minimal interruption.

To enable the automatic client reroute feature, issue the `UPDATE ALTERNATE SERVER` command on the primary database. Here is the syntax of the command:

```
UPDATE ALTERNATE SERVER FOR DATABASE database-alias
USING HOSTNAME alternate-server-hostname PORT port-number
```

To enable the client reroute feature on the TEST1 database on server1 so that all existing and new client connections are rerouted to server2 following a takeover, issue the following command on server1:

```
UPDATE ALTERNATE SERVER FOR DATABASE test1 USING
HOSTNAME server2.torolab.ibm.com PORT 50000
```

In this example, 50000 is the SVCENAME port, not the HADR port.

The alternate server information is stored in server1's database directory. If you issue a `LIST DB DIRECTORY` command, notice that the alternate server information (both the hostname and the port number) has been added:

```
Database alias           = TEST1
Database name           = TEST1
Local database directory = C:
Database release level  = b.00
Comment                 =
Directory entry type    = Indirect
Catalog database partition number = 0
Alternate server hostname = server2.torolab.ibm.com
Alternate server port number = 50000
```

When a remote client application connects to this database, the alternate server information is retrieved and gets stored in the database directory of the client as well. Using this information, the client knows which server to connect to in case the primary server, server1, becomes unavailable.

Because the client does not get this alternate server information until it makes a connection to the primary server, issue the `ALTERNATE SERVER` command before a takeover operation and make sure the client has at least made one successful connection to it prior to the takeover.

Performing a takeover with the automatic client reroute feature enabled

Now let's look at an example of how a takeover works on database TEST1 with the automatic client reroute feature enabled.

You have set up an HADR environment for the TEST1 database using the five steps discussed in the Initializing HADR section. You have also run the `UPDATE ALTERNATE SERVER` command on server1 to specify server2 as its alternate server.

To demonstrate how the client reroute feature works in an HADR environment, make a remote client connection to TEST1 on server1 and then perform a takeover on the standby database server, server2. If HADR and the client reroute feature work properly, you should be able to see that the client connection is automatically rerouted to server2 after the takeover.

In this example, perform a takeover without the `BY FORCE` option.

First, make a remote client connection to the TEST1 database on server1. Do this by simply issuing a `CONNECT TO DATABASE` command on a remote DB2 client:

```
CONNECT TO DB test1 USER user1
```

To make such a connection, the client must have already been configured to connect to server1. Please refer to the [first tutorial in this series](#) for information on how to configure a remote client connection.

After the connection is made, the alternate server information on the primary server is retrieved and stored in the client's database directory:

```
Database alias           = TEST1
Database name           = TEST1
Node name                = server1
Database release level  = b.00
Comment                 =
Directory entry type    = Remote
Catalog database partition number = -1
Alternate server hostname = server2.torolab.ibm.com
Alternate server port number = 50000
```

Make sure the connection is successfully established and that you can query the TEST1 database. Issue a `LIST APPLICATION` command on server1, it should show the remote client connection on database TEST1:

```
D:\Programs\IBM\SQLLIB\BIN>db2 list applications

Auth Id  Application      Appl.   Application Id      DB      # of
-----  -----
USER1    db2bp.exe       8       9.26.94.149.24336.060601000305
                                         TEST1  1
```

Next, perform a takeover on the TEST1 database. Issue the following command on the standby server, server2:

```
TAKEOVER HADR ON DB test1
```

Let's see what happens to the remote client connection following the takeover.

Try to query the TEST1 database again from the client. Any query to the TEST1 database should return an error message. For example, issue a LIST TABLES command from the client connection:

```
D:\Programs\IBM\SQLLIB\BIN>db2 list tables
SQL30108N A connection failed but has been re-established.
The hostname or IP address is "server2" and the service name or port number is "50000".
Special registers may or may not be re-attempted (Reason code = "1").
SQLSTATE=08506
```

This message tells us that the existing connection to server1 has failed. However, it has been reestablished by connecting to a server whose hostname is 'server2', which is our standby server.

This proves that the client reroute feature worked. You are now connected to the new primary server, server2.

Issue a LIST APPLICATIONS command on server2. You see that the remote client connection has been rerouted there:

```
H:\Programs\IBM\SQLLIB\BIN>db2 list applications
```

Auth Id	Application Name	Appl. Handle	Application Id	DB Name	# of Agents
USER1	db2bp.exe	55	9.26.94.149.29200.060601001803	TEST1	1

If you query the database again, you get a response from the new primary server. Also, any new connections to the old primary server are rerouted to the new primary server as well.

Get the HADR status of a database using the GET SNAPSHOT command. Issue the following command on server2 and you should see that server2 has taken on the primary database role:

```
db2 "get snapshot for database on test1"

HADR Status
Role                = Primary
State               = Peer
Synchronization mode = Nearsync
Connection status   = Connected , 05/31/2006 20:17:24.989618
Heartbeats missed   = 0
Local host          = server2.torolab.ibm.com
Local service       = 80000
Remote host         = server1.torolab.ibm.com
Remote service      = 70000
Remote instance     = db2inst1
timeout(seconds)    = 120
Primary log position(file, page, LSN) = S0000007.LOG, 66, 0000000002FE2D9B
```

```
Standby log position(file, page, LSN) = S0000007.LOG, 66, 0000000002FE2D9B  
Log gap running average(bytes) = 0
```

Reintegrating a database after a takeover operation

After a takeover operation is performed, if the failed database later becomes available, it does NOT automatically assume the role of a primary database.

Return it to its status as primary database by doing the following:

1. Repair the system where the original primary database resided. This could involve repairing failed hardware or rebooting the crashed operating system.
2. Restart the failed primary database as a standby database.
3. Perform a takeover operation to take over the primary database role.

Reintegration fails if the two copies of the database have incompatible log streams. In particular, HADR requires that the original primary database did not apply any logged operation that was never reflected on the original standby database before it took over as the new primary database. If this did occur, restart the original primary database as a standby database by restoring a backup image of the new primary database or by initializing a split mirror.

After the original primary database has rejoined the HADR pair as the standby database, perform a takeover operation to switch the roles of the databases to enable the original primary database to be the primary database once again.

Stopping HADR

Use the `STOP HADR` command to stop HADR operations on the primary or standby database. Stop HADR on one or both of the databases. If you are performing maintenance on the standby system, you only need to stop HADR on the standby database. To stop using HADR completely, stop HADR on both databases

If you want to stop the specified database but still want it to maintain its role as either an HADR primary or standby database, do not issue the `STOP HADR` command. If you do, the database becomes a standard database and may require reinitialization to resume operations as an HADR database. Instead, issue the `DEACTIVATE DATABASE` command.

To stop HADR on the TEST1 database, issue:

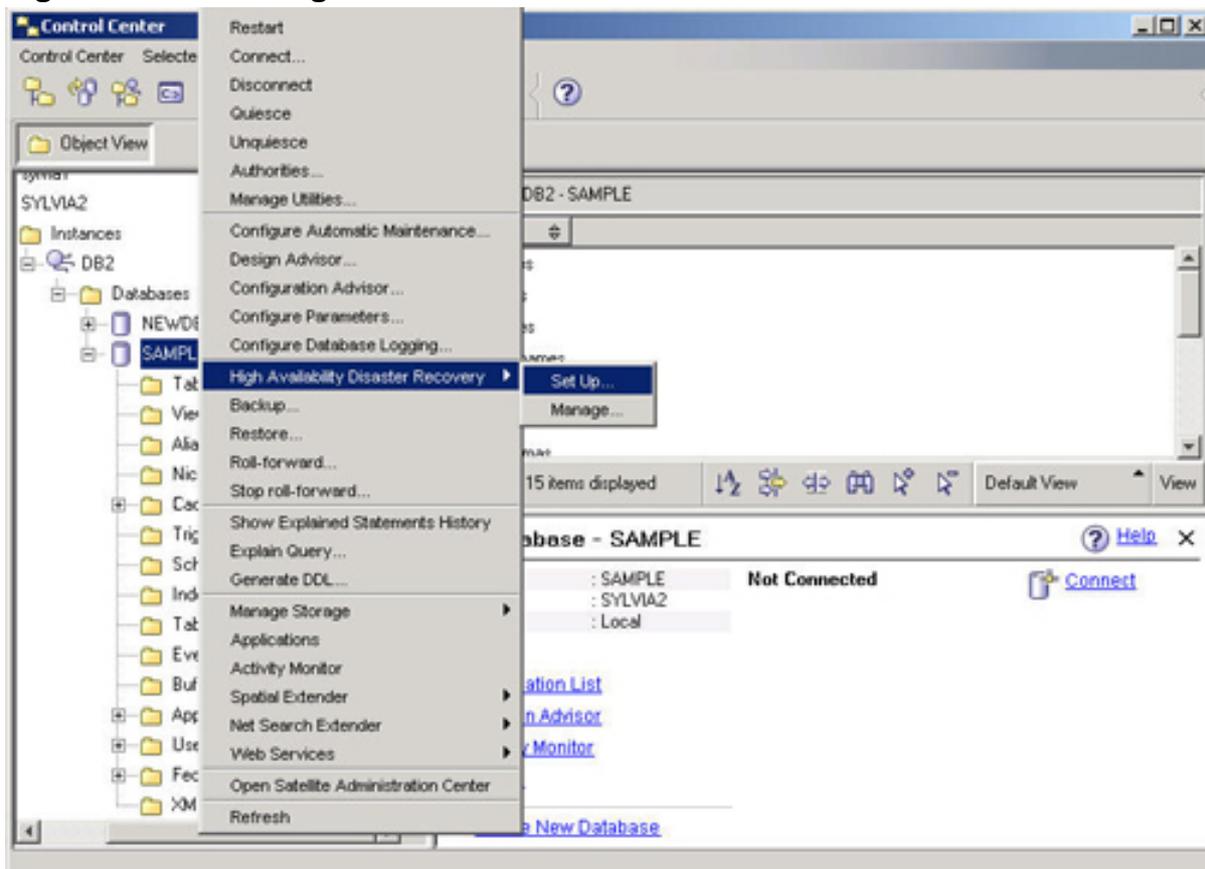
```
STOP HADR ON DB test1
```

The HADR wizard

As you can see, HADR is a powerful feature used to implement a high availability solution. A graphical user interface tool called the HADR wizard is available from the DB2 Control Center to help you set up and configure HADR.

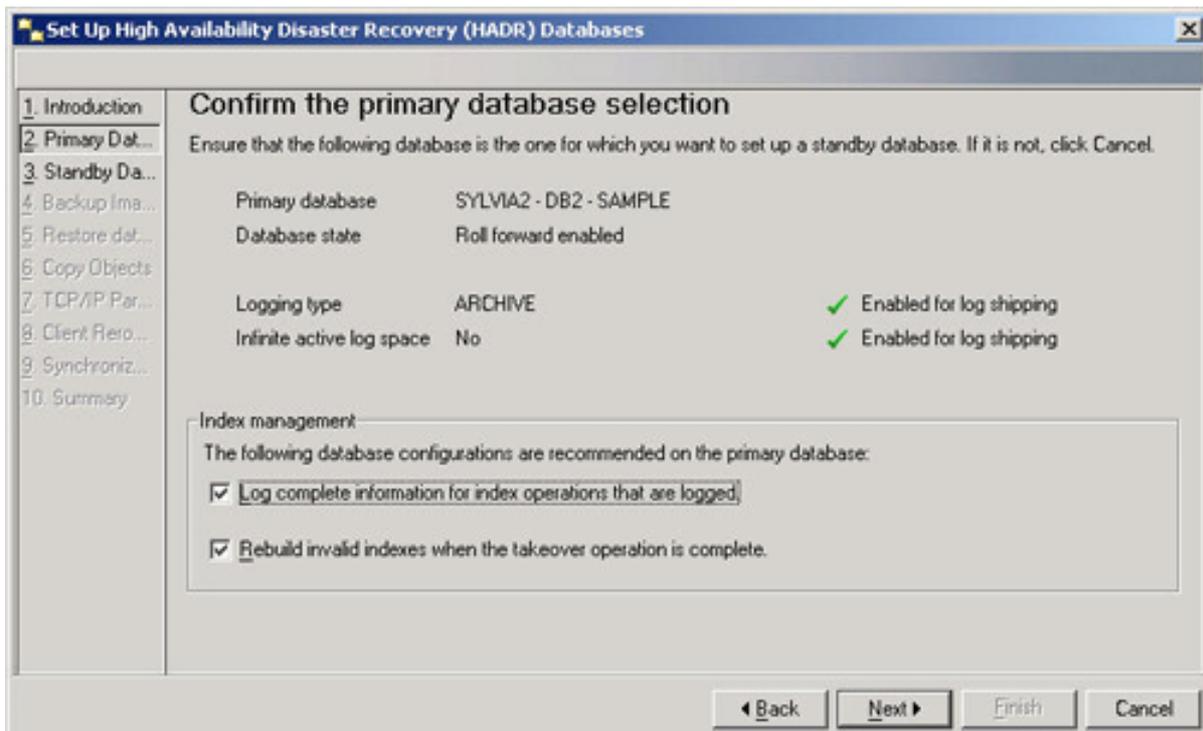
The wizard guides you through the tasks required to set up the HADR environment, stop and start HADR, and switch database roles under HADR. To launch the wizard, go to the Control Center and right-click on the database. Select **High Availability Disaster Recovery**. Choose to set up or manage HADR:

Figure 14. Launching the HADR wizard from the Control Center



If you choose **Set Up**, a step-by-step wizard is launched as demonstrated in Figure 14.

Figure 15. Setting up the HADR environment



If you use the Control Center to set up an HADR environment, the automatic client reroute feature is automatically enabled.

Section 6. Dynamic configuration parameters

Dynamic database configuration parameters

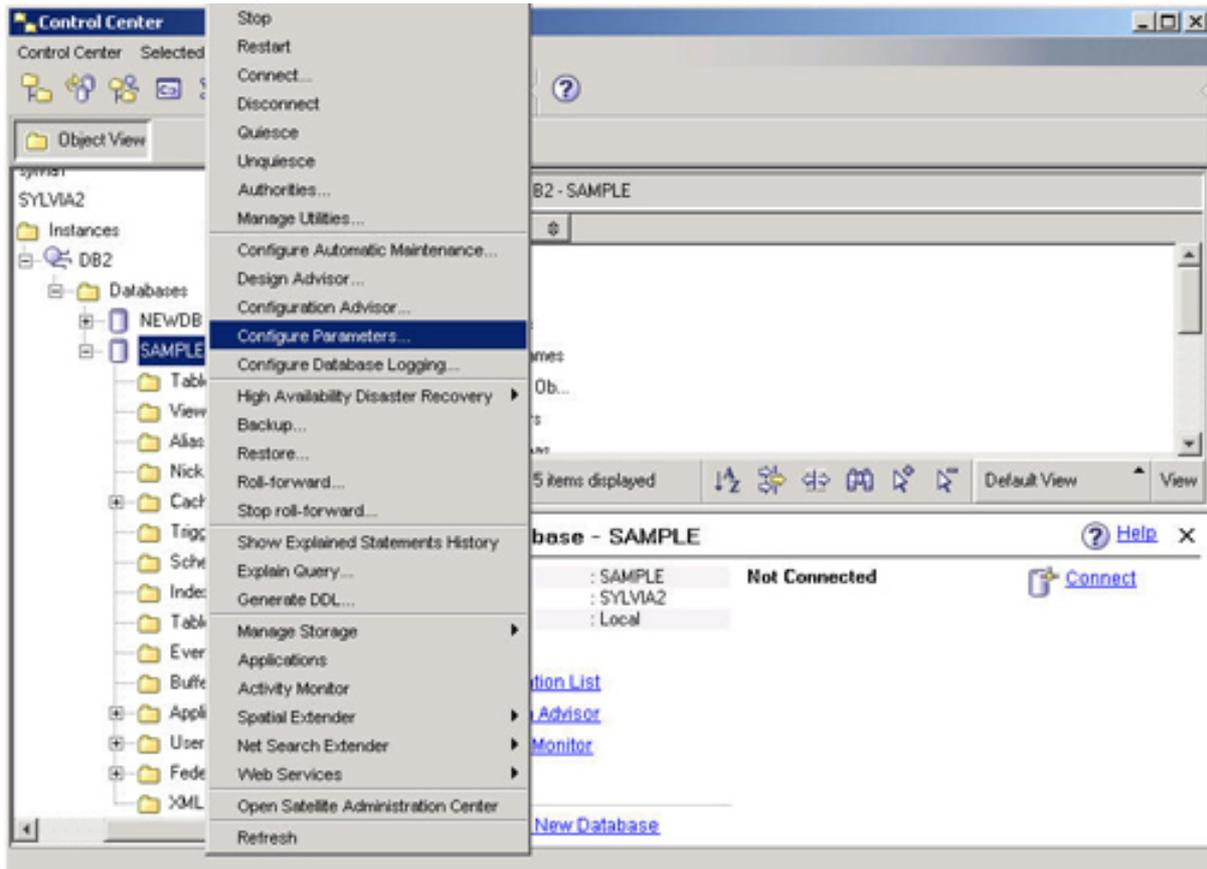
The ability to update database configuration parameters, without having to recycle the database, is an important aspect of an HA system.

Dynamic database configuration operations are performed while users are connected to the database. In other words, the database is online while you update your system. The new configuration values take effect immediately; you do not have to restart the database.

For example, if you discovered the `logsecond` parameter is set to a value too low for your environment, you would want to increase it. Because `logsecond` is a dynamic configurable parameter, you can alter its value while users are connected and the value takes effect immediately. You do not need to disconnect your users to make this parameter change.

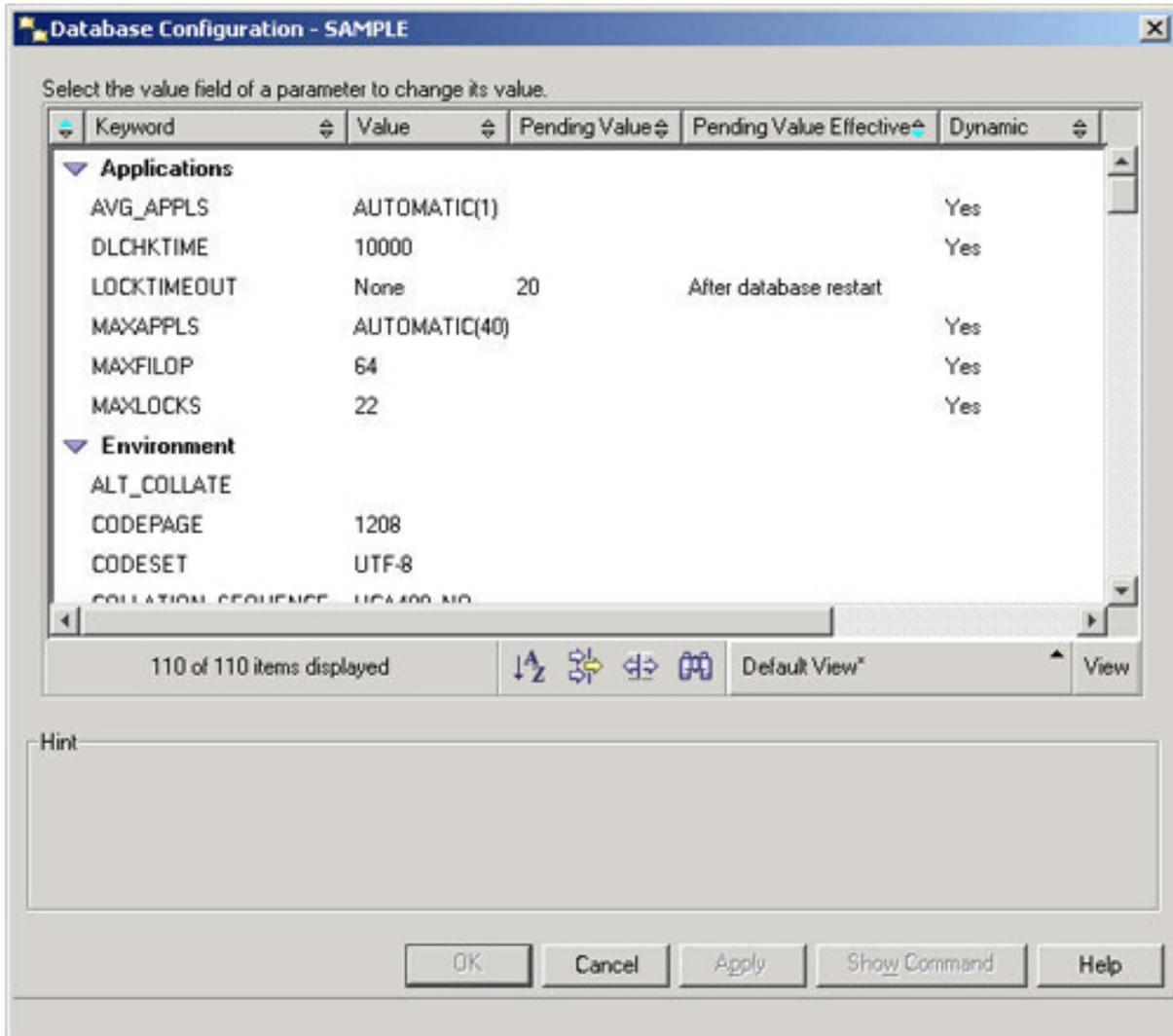
Not all DB2 parameters are dynamically configurable. To find out whether a parameter is dynamic, use the Control Center:

Figure 16. Configure database parameters using the Control Center



If you right-click on the database name from the Control Center and select **Configure Parameters**, you get a list of the database configuration parameters, as shown in the Figure 17:

Figure 17. Control Center showing database parameters



The value in the Dynamic column indicates whether a parameter is dynamically configurable or not. If a parameter is not dynamic and you update its value, the new value shows up in the Pending Value column. The value takes effect when the database is restarted.

You might also get this information using the `GET DB CFG` command with the `SHOW DETAIL` option. For example:

```
db2 "get db cfg for sample show detail"

      Database Configuration for Database sample

Description                Parameter                Current Value    Delayed Value
-----
Database configuration release level = 0x0b00
Database release level         = 0x0b00

Database territory            = C
Database code page            = 1208
Database code set              = UTF-8
...
Interval for checking deadlock (ms) (DLCHKTIME) = 10000          10000
Lock timeout (sec)             (LOCKTIMEOUT)    = -1             20
...
```

The Delayed Value column shows the value that is to take effect when the database is restarted. If this value does not match the current value, then you know that the parameter is not dynamic.

Dynamic database manager configuration parameters

Like the dynamic database configuration parameters, some database manager configuration parameters are also dynamic. You may use either the Control Center or the `GET DBM CFG SHOW DETAIL` command to find these parameters.

Dynamic database manager configuration operations are performed while an application is attached to the instance.

For parameters that are not dynamic, the reconfigured values take effect when the instance is restarted.

Section 7. Summary

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

- The concept of high availability and the three characteristics of a high availability system
- How log shipping works and how to set up log shipping between a primary database and a standby database
- DB2 support for split mirror support and suspended I/O
- How to use the `db2inidb` command to initialize a split mirror
- How HADR works
- How to initialize an HADR system, perform a takeover operation, reintegrate a database back into the HADR system, and stop HADR
- How to use the client automatic reroute feature in an HADR environment to minimize interruptions to client applications
- Dynamically configurable database configuration parameters and database manager configuration parameters.

Congratulations! You have just completed the final tutorial in the [DB2 9 Database Administration 731 certification prep series](#) for DB2 9 for Linux, UNIX, and Windows Database Administration (Exam 731). You should be prepared to take the exam with confidence. Good luck!

Resources

Learn

- Check out the other parts of the [DB2 9 DBA exam 731 prep tutorial series](#).
- Learn more about managing DB2 from the following resources:
 - *DB2 9 Administration Guide: Performance*
 - [DB2 9 Information Center](#), the following topics:
 - Setting up the DB2 server after manual installation
 - Configuring client to server communications
 - Configuring DB2 server communications
 - *DB2 9 Administration Guide: Implementation*
- For more information on the DB2 9 for Linux, UNIX, and Windows Database Administration Certification (Exam 731), check out these links:
 - [Information Management Training information](#)
 - [General certification information](#) -- including some book suggestions, exam objectives, and courses.
- [DB2 9 overview](#). Find information about the new data server that includes patented pureXML technology.
- [Certification exam](#) site. Click the exam number to see more information about Exams 731.

Get products and technologies

- A [trial version of DB2 9](#) is available for free download.
- Download [DB2 Express-C](#), a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.

Discuss

- [Participate in the discussion forum for this content](#).

About the authors

Michael Dang

Michael D. T. Dang currently works for IBM Global Services as a Senior DB2 UDB Database Administrator. He has worked at IBM for eight years. Michael is an IBM certified DB2 UDB administrator and application developer and is an active contributing author to IBM developerWorks.

Sylvia Qi

Sylvia F. Qi works at the IBM Toronto Lab as a WebSphere Application Server function verification tester. Previously, Sylvia worked as a DB2 UDB Level 2 support engineer for six years. She has in-depth knowledge in all areas of DB2 UDB, and is an IBM certified DB2 UDB administrator and application developer. Sylvia holds a bachelor of Computer Science degree from McGill University. She is an active contributing author of IBM developerWorks. Sylvia has recently co-authored the book *Understanding DB2 - Learning Visually with Examples* (ISBN 0131859161).