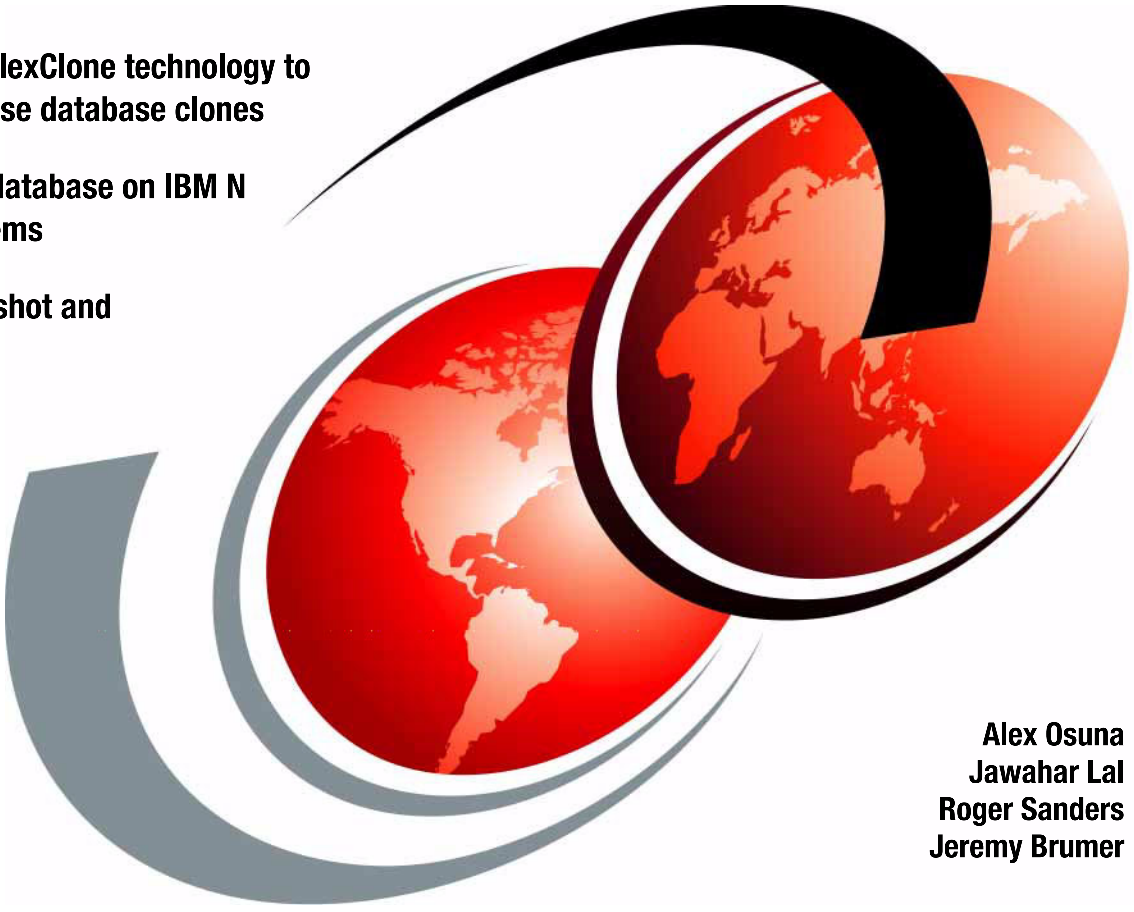
Information Management software

Using IBM DB2 UDB with IBM System Storage N series

Exploiting FlexClone technology to build and use database clones

Creating a database on IBM N series systems

Using Snapshot and SnapMirror



Alex Osuna
Jawahar Lal
Roger Sanders
Jeremy Brumer

Redbooks

ibm.com/redbooks

International Technical Support Organization

Using IBM DB2 UDB with IBM System Storage N series

December 2006

SG24-7323-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (December 2006)

This edition applies to the IBM System Storage N series and Data ONTAP Version 7.

Contents

Notices

Trademarks

Preface

Database cloning is the process by which you create an exact copy of a DB2® database, either by physically copying the data or by performing what is known as a *redirected restore*.

Database cloning is performed frequently by database administrators to provide near-production data for various business needs such as application development, QA testing, and report generation. Traditional methods of cloning a database pose various challenges, including system downtime and degraded system performance during the cloning process. Additionally, a large amount of storage space is required to store each clone. Furthermore, the maintenance overhead can be enormous if each cloned database requires a frequent data refresh.

This IBM® Redbook describes the process used to create a clone of an IBM DB2 UDB database using FlexClone technology. This book also covers creating a database clone on a disaster recovery site that has replicated data using Data ONTAP® SnapMirror® technology.

This book is meant for database administrators, database developers, database designers, and storage administrators.

The team that wrote this redbook

This IBM Redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.

Alex Osuna is a Project Leader at the ITSO, Tucson, Arizona. He writes extensively and teaches IBM classes worldwide on all areas of storage. Before joining the ITSO, he was a Systems Engineer with Tivoli®. He has been involved with storage for over 26 years and the IT industry for over 28 years in areas of service, planning, early ship programs, Advanced Technical Support, and Systems Engineering. He holds over 10 certifications from IBM, Microsoft®, and Red Hat.

Jawarhar Lal is an Alliance engineer for IBM DB2 at Network Appliance™ Inc., RTP, North Carolina. He holds a Masters degree in Computer Science from University of Rajasthan, Jaipur (India). Currently, he is working on his MBA degree from Kenan Flagler Business School, UNC Chapel Hill, North Carolina.

He has been involved in storage and IT industry over 11 years, in areas of database programming, modeling, designing, administration, performance tuning and storage integration. He writes extensively on storage and databases integration. He is an IBM DB2 UDB Certified DBA and holds four other certifications from IBM and Oracle.

Roger Sanders is a Senior Manager with IBM Alliance Engineering with Network Appliance, Inc. He has been designing and developing databases and database applications for more than 20 years and has been working with DB2 Universal Database™ and its predecessors since it was first introduced on the IBM PC (as part of OS/2® 1.3 Extended Edition). He has written articles for IDUG Solutions Journal and Certification Magazine, authored DB2 tutorials for the IBM developerWorks® Web site, presented at several International DB2 User's Group (IDUG) and regional DB2 User's Group (RUG) conferences, taught classes on DB2 Fundamentals and Database Administration (DB2 8.1 for Linux®, UNIX®, and Windows®), and is the author of nine books on DB2 and one book on ODBC. Roger is also a member of the DB2 Certification Exam development team and the author of a regular column (Distributed DBA) in *DB2 Magazine*.

Jeremy Brumer has been working with the DB2 system verification test department in the IBM Toronto Lab for the last six years. His areas of expertise include high availability solutions and DB2's integration with N-series.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Introduction to FlexClone

Database *cloning* is a process by which you can create an exact copy of a DB2 database. You can either physically copy the data or perform a *redirected restore*. Database cloning is performed frequently by database administrators to provide near-production quality data for various business needs such as application development, QA testing, and report generation. Traditional methods of cloning a database pose various challenges, including system downtime and degraded system performance during the cloning process illustrated in Figure 1-1. Additionally, a large amount of storage space is required to store each clone. Furthermore, the maintenance overhead can be enormous if each cloned database requires a frequent data refresh.

In this era of high system availability, organizations cannot afford extended downtime and degraded performance for their production systems. Therefore, the ability to create a usable database clone quickly and with virtually no impact on the production system is extremely important.

FlexClone is an advanced and proven technology that helps database and system administrators deliver a near-instantaneous, point-in-time copy of the production database. The database cloning process is completed in a few seconds, with virtually no performance impact on the production system. The cloned database is similar to the production database in all aspects. However, unlike a traditional database clone, a FlexClone database consumes no extra disk space at the time of creation.

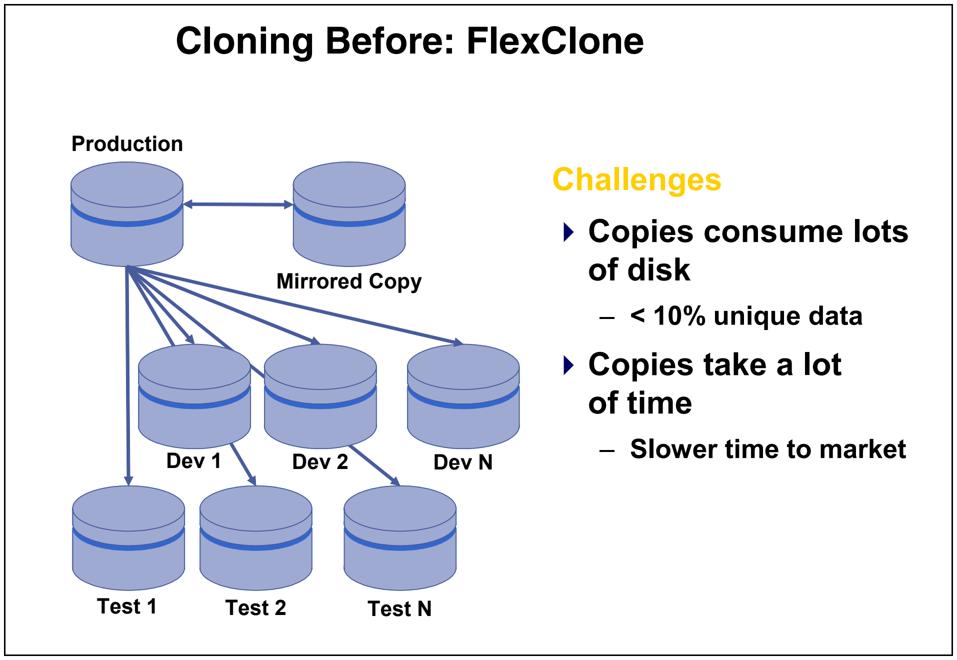


Figure 1-1 Cloning before FlexClone

1.1 Purpose and Scope

This book describes the process used to create a clone of an IBM DB2 UDB database using FlexClone technology. It also covers creating a database clone on a disaster recovery site that has replicated data using Data ONTAP SnapMirror technology. Specifically, the following topics are covered in this document:

- Creating an IBM DB2 UDB database on a IBM N Series storage system.
- Creating a clone of an offline DB2 UDB database on same aggregate.
- Creating a clone of an online DB2 UDB database on same aggregate.
- Creating a clone of an offline DB2 UDB database on a remote (disaster recovery) site.
- Creating a clone of an online DB2 UDB database on a remote (disaster recovery) site

1.2 Overview of FlexClone technology

FlexClone, depicted in Figure 1-2, is a powerful new feature that adds a new level of agility and efficiency to storage operations by allowing an individual to create an instant clone of a flexible volume (FlexVol volume). A FlexClone volume is a writable point-in-time image of a FlexVol volume or another FlexClone volume. With FlexClone, it takes only a few seconds to create a clone of a *FlexVol* volume, the original production volume. Such a volume can be created without interrupting access to the parent volume upon which the clone is based. The clone volume uses space very efficiently, allowing both the original FlexVol volume and the FlexClone volume to share common data, storing only the data that changes between the original volume and the clone. This provides a huge potential saving in storage space, resources, and cost. In addition, a FlexClone volume has all the features and capabilities of a regular FlexVol volume, including the ability to be grown or shrunk and the ability to be the source of another FlexClone volume. A database clone can be created simply by creating a clone of the FlexVol volumes that are used for the database's data and transaction logs.

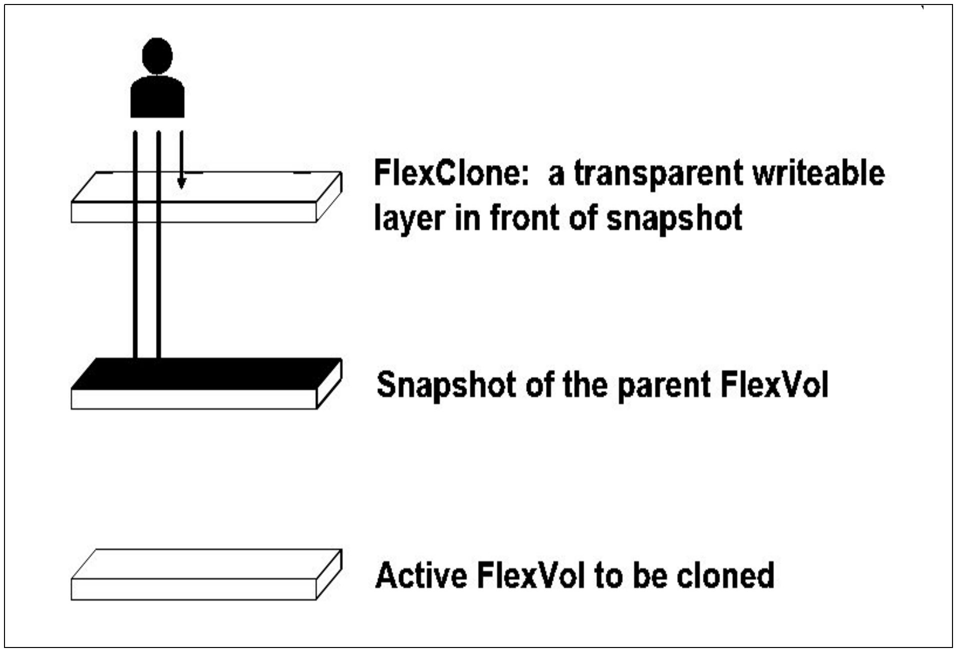

FlexClone: a transparent writeable layer in front of snapshot
Snapshot of the parent FlexVol
Active FlexVol to be cloned

Figure 1-2 FlexClone

1.3 Overview of the SnapMirror technology

Data ONTAP SnapMirror technology provides an efficient way to transfer data from one IBM N Series storage system to another, as in Figure 1-3. The storage system from which data is transferred is called the SnapMirror *source*, and the storage system to which data is transferred is called the SnapMirror *destination*. A SnapMirror source and its corresponding destination can reside on the same storage system or on two separate storage systems that are miles apart, provided that both storage systems are able to communicate with each other over a network. Data changes made at the SnapMirror source are replicated continuously at the SnapMirror destination to keep the data at both locations synchronized.

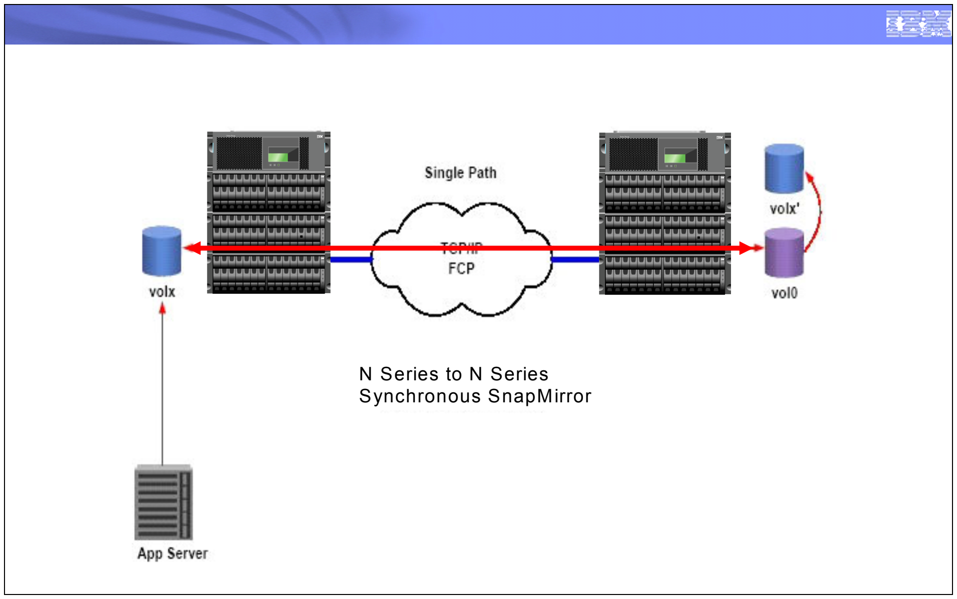


Figure 1-3 SnapMirror technology in action

Data ONTAP provides both asynchronous and synchronous SnapMirror functionality. With asynchronous SnapMirror, each write is acknowledged as soon as it is written to NVRAM at the source, even if the destination has not yet received and processed the request. Normally data changes at the SnapMirror source are copied to the destination periodically, based on the schedule defined in the /etc/snapmirror.conf file that resides on the destination storage system.

With synchronous SnapMirror, each time a transaction attempts to write data to disk, the data is sent to both the SnapMirror source and the SnapMirror destination in parallel. Not until both storage systems have committed the write to

NVRAM does the system acknowledge that the transaction is complete. In other words, the application that initiated the write must wait until it receives the acknowledgement from both the source and destination storage systems before it can continue.

A detailed discussion of SnapMirror is beyond the scope of this book. For further information about SnapMirror technology, refer to the IBM Redpaper *N Series Snapshot: a Technical Discussion*, REDP-4132.

http://www.redbooks.ibm.com/abstracts/redp4132.html?Open

1.4 Ensuring consistency for DB2 UDB database (suspended I/O)

To obtain a consistent Snapshot™ copy for an online recoverable DB2 UDB database, all write operations must be suspended before invoking the Data ONTAP snapshot command. IBM DB2 UDB 7.1 (FixPak 2) and later provides support for temporarily suspending writes to the database by executing the SET WRITE SUSPEND command. This functionality allows an individual to suspend temporarily all writes to the database and its transaction log files before creating a Snapshot copy. Read-only transactions are able to continue running against the database while it is in write-suspended state. However, some transactions may be forced to wait if they require disk I/O for example, to flush dirty pages from the buffer pool or to flush log records from the log buffer. These transactions proceed normally when write operations on the database are resumed.

To suspend write operations to a database, connect to the database and execute the following command:

```
set write suspend for database
```

After you create Snapshot copies of the FlexVol volumes using the snapshot command, resume write operations to a database by executing the following command on the database server:

```
set write resume for database
```

1.5 The db2inidb command

Along with the SET WRITE command, the **db2inidb** command was introduced in DB2 UDB 7.1 FixPak 2. This command is used primarily to initialize a snapshot copy of a database; in the context of creating a clone of a DB2 UDB database from a Snapshot copy, the purpose of **db2inidb** is to initialize the clone database as a snapshot image. The syntax for the **db2inidb** command is as follows:

```
db2inidb [DatabaseAlias] as [snapshot|standby|mirror] <relocate using [ConfigFile]>
```

Where the following variables are defined as:

- `Database` Alias identifies the alias assigned to the snapshot (secondary) database that is to be initialized.
- `ConfigFile` identifies a configuration file that contains information about how the database files contained in the snapshot copy of the database are to be relocated when the snapshot image is initialized.

Note: Parameters shown in angle brackets (< >) are optional. Parameters or options shown in square brackets ([ ]) are required and must be provided. A comma followed by ellipses (…) indicates that the preceding parameter can be repeated multiple times.

The **db2inidb** command can also be used to change that database name, database instance name, and tablespace header information based on the user provided configuration file. The configuration file must adhere to the format in Example 1-1.

Example 1-1 Configuration file

```
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
-----
STORAGE_PATH= oldStoragePath1,newStoragePath1
STIRAGE_PATH= oldStoragePath2,newStoragePath2
```

The sample configuration file looks similar to Example 1-2.

Example 1-2 Sample configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

Note: It is important to note that if the database has automatic storage enabled, you must specify changes to the location of the database storage paths (using the STORAGE_PATH parameter), not the tablespace containers (using the CONT_PATH parameter).

For example, to initialize a DB2 database named mydbcl, you would execute the following command at the database server:

```
db2inidb mydbcl as snapshot
```

A second example, to initialize a clone DB2 database named mydbcl, and change the database name, database instance name, and tablespace header information using configuration file named config.txt, you would execute the following command at the database server:

```
db2inidb mydbcl as snapshot relocate using config.txt
```

It is important to note that **db2inidb** is never run on the production or the parent database. Instead, it must only be executed against the clone database.

1.6 The db2relocatedb command

The **db2relocatedb** command allows a database administrator (DBA) to change the location of one or more tablespace containers or an entire database, without having to perform a backup and a redirected restore operation. It also provides a way to rename a database and change the instance to which a database belongs, per specifications in a configuration file that is provided by the user.

When executed, this command makes the necessary changes to the DB2 instance and the appropriate database support files. From the clone database perspective, it is used to rename the clone database, change the DB2 instance with which the clone is associated, and change the tablespace container metadata for the tablespace containers that are associated with the clone.

To rename a clone database and update the metadata for its tablespace containers, you would execute the following command on the database server:

```
db2relocatedb -f [ConfigFile]
```

Where `ConfigFile` identifies the name of a configuration file that contains information that is needed to alter the DB2-specific metadata stored in files associated with a database.

The configuration file used must follow the same format specified in 1.5, “The db2inidb command” on page 6.

For example, to relocate a DB2 database using a configuration file named config.txt, you would execute the following command on the database server:

```
db2relocatedb -f config.txt
```

1.7 Advantages of cloning a database using N series FlexClone technology

There are abundant business needs that require production or near-production quality data. These needs can be satisfied by creating a clone of the production database. The added overhead and the complexities associated with the traditional method of creating a clone database pose immense challenges for database and storage administrators. However, N series FlexClone technology has made the database cloning process so simple and easy that a clone can be created in couple of seconds without creating any system overhead. The clone database created using the N series FlexClone feature resides on the same storage system as the parent database.

The following list describes some of the key business requirements that can benefit greatly from a clone database created using FlexClone technology:

- Writable Disaster Recovery Destination: Generally, a disaster recovery site is equipped with systems that have almost the same capacity and power as the production systems and is not accessible until disaster strikes.

  The capacity and power of the disaster recovery site systems can be used for various purposes such as reporting, data mining, testing, and development by making the replicated data accessible without affecting the data replication process.

- Reporting Environment: Business reports need read-only access to the production database. Normally, reporting environments have near-production quality data and are refreshed based on a frequency, every night, for example. By cloning a disaster recovery site, you can deliver a highly-efficient and low-cost reporting environment.

- Database Software and Application Upgrade Test: Applications upgrades and database software patches can be tested on the cloned database before they are implemented into production.
- Data Mining: Data-mining software and operations can be implemented with great flexibility because both reads and writes are allowed on a cloned database.
- Data Warehouse/Data Mart: In typical data warehouse architecture, data from operational data stores (ODS) are copied to staging areas. The staging area is used for performing cleansing and transformation, and as the data source for data load into the data warehouse database.

  By creating a clone of the replicated data at the disaster recovery site or at a production database, you can provide a staging area that can be refreshed quickly and can be used for cleansing and transformation of the data.
- Standby Database: Immediately resume read-write workload on discovering corruption in the production dataset by mounting the clone instead. You need to use database features such as DB2 write-suspend mode to prepare the database volumes transparently for cloning by delaying write activity to the database. This delay is necessary because databases must maintain a point of consistency.
- Application Testing: Enterprise production data is growing exponentially and the applications testing is not performed on near-production data because of the size of the data and maintenance overhead. Therefore, some real-life system problems go undetected during the testing phase of the system. By providing a cloned database, you can deliver near-production data for application testing and avoid production issues.
- Online Database Backup: The clone database is a picture image of the database file system at the time the clone was created. If necessary, the primary database can be restored from the system picture created for the clone. Alternatively, applications can point directly to the clone database.

1.8 Configuration used for this publication

The information presented in this book has been tested only with a specific hardware and software configuration. Network Appliance (NetApp®) and IBM have tested IBM DB2 UDB V8.2 for Linux. Hosts used for these tests were running RHEL 4.0. NetApp has not tested this configuration with other combinations of hardware and UNIX flavors. There can be significant differences in your environment configuration and setup that will alter the procedures necessary to accomplish the objective outlined in this book. If you find that any of the procedures described in this paper do not work in your environment, please contact this author immediately.

2

Preparation

This chapter describes the set up work that we performed before starting the tests. This setup includes assumptions about the conditions of the operating environment and the equipment.

2.1 General assumptions

In order to take maximum advantage of the procedures and steps described in this publication, it is assumed that you are familiar with the following:

- Commands and operations of Data ONTAP and the IBM N series storage system
- Administration and operations of an IBM DB2 UDB instance and database as well DB2 utilities such as db2inidb and db2relocatedb
- UNIX system administration commands
- Understanding of file and block access protocols such as Fibre Channel and iSCSI

It is assumed that the IBM N Series storage systems used are licensed for NFS, FCP, iSCSI, FlexClone, and SnapMirror. Additionally, license keys for SnapMirror sync are required for both source and destination storage systems, if synchronous SnapMirror is to be used.

It is also assumed that the UNIX hosts used to access the production database and the cloned database have the following software and utilities installed:

- IBMM DB2 UDB ESE V8
- For a SAN environment, a supported HBA, SanSurfer utility, and host attach kit are installed and configured. Host attach kits can be downloaded from the from the IBM Web site:

  `http://www-03.ibm.com/servers/storage/support/allproducts/downloading.html`

2.2 Environment assumptions

This technical report covers cloning of an IBM DB2 UDB Enterprise Server Edition V8 database running under UNIX with database storage on a IBM N Series storage system. The sample scripts and steps in this publication assume the following:

- The name of the storage system containing the production database is *srcstore*.
- The name of the storage system used as the SnapMirror destination is *dststore*.
- The database host system used to access the production database is *hostsrc*.
- The database host system used to access the clone database is *hostdst*.

- The name of the source database is *mydb*.
- The name of the cloned database is *mydbcl*.
- The name of the aggregate on the storage systems is *dbaggr*.
- The name of the FlexVol volume used to store the production database table data is *dbdata*.
- The name of the FlexVol volume used to store the production database transaction logs is *dblogs*.
- The name of the clone volume that is created from the FlexVol volume named dbdata is *dbdata_cl*.
- The name of the clone volume that is created from the FlexVol volume named dblogs is *dblogs_cl*.
- The mount points used for the production database are /mnt/dbdata and /mnt/dblogs.
- The mount points used to mount the volumes used for the clone database are /mnt/dbdata_cl and /mnt/dblogs_cl.

The scripts contained in this document might require significant modifications to run under your version of UNIX.

2.3 Security and access issues

You need to make sure that each FlexVol volume used for the DB2 UDB database's data and transaction logs have their security style set to UNIX. The security style can be updated by executing the following command on the storage system:

```
qtree security [FlexVolPath] unix
```

Where FlexVolPath identifies the flexible volume path on the storage system that is used for the database.

For example, to update the security style of a FlexVol volume named dbdata, you would execute the following command on the storage system:

```
qtree security /vol/db2data unix
```

2.4 Network and storage infrastructure

Figure 2-1 illustrates a simple basic architecture used to create a clone of a DB2 UDB database in the UNIX, NetApp FAS, or IBM N Series storage system environment.

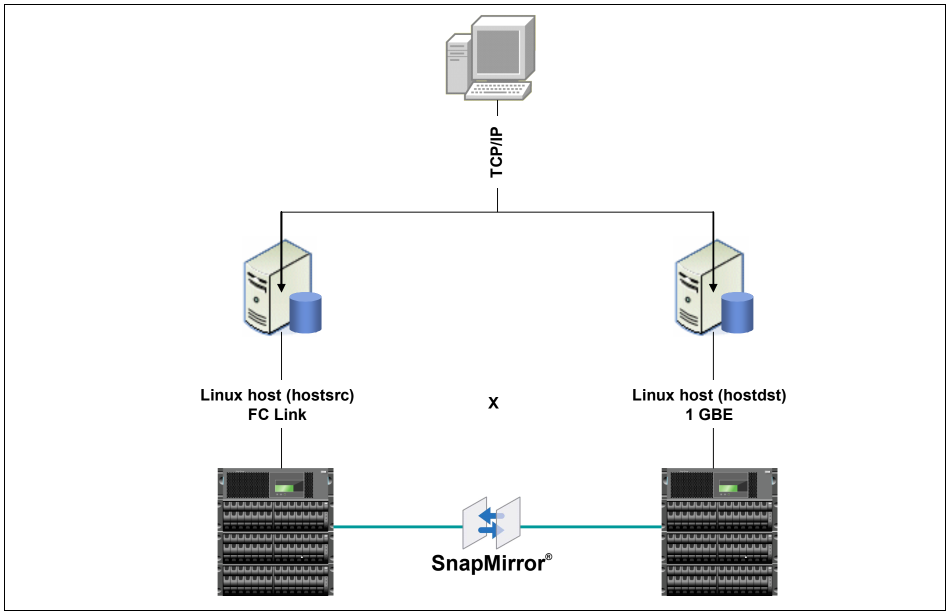


Figure 2-1 Basic architecture

A FlexClone volume of a FlexVol volume, also known as FlexClone volume, resides in the same aggregate as the parent FlexVol volume. Figure 2-2 illustrates that both the parent volumes and the clone volumes reside in the aggregate named dbaggr.

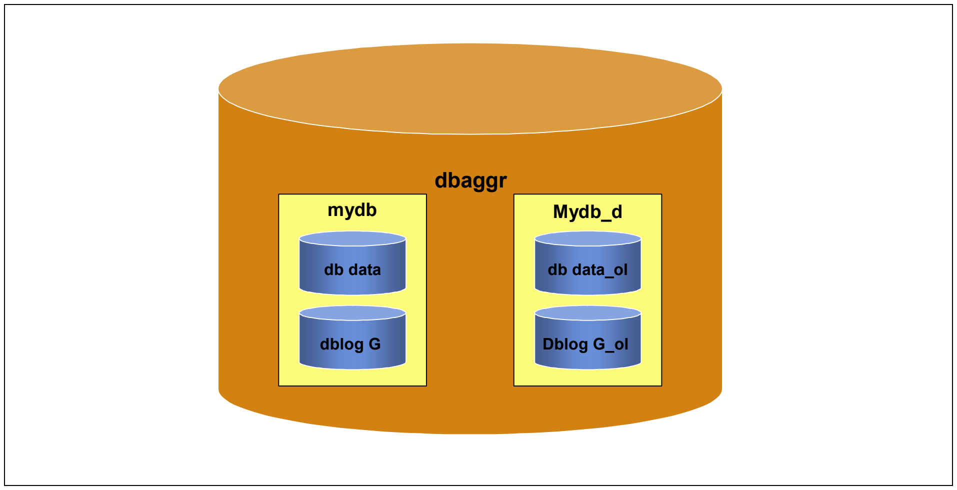


Figure 2-2 Both the source and clone volumes reside in the same aggregate

The FlexClone feature, combined with SnapMirror, enables database and storage administrators to create a clone on another aggregate or on another storage system.

Figure 2-3 illustrates the use of SnapMirror to replicate the data from the source volume to the destination volume, creating a clone on the destination storage system.

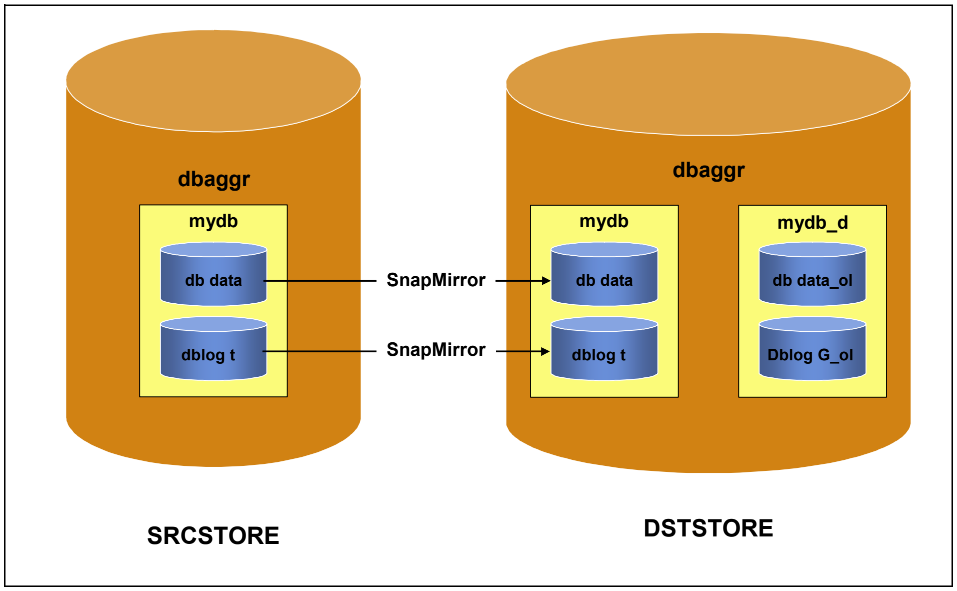
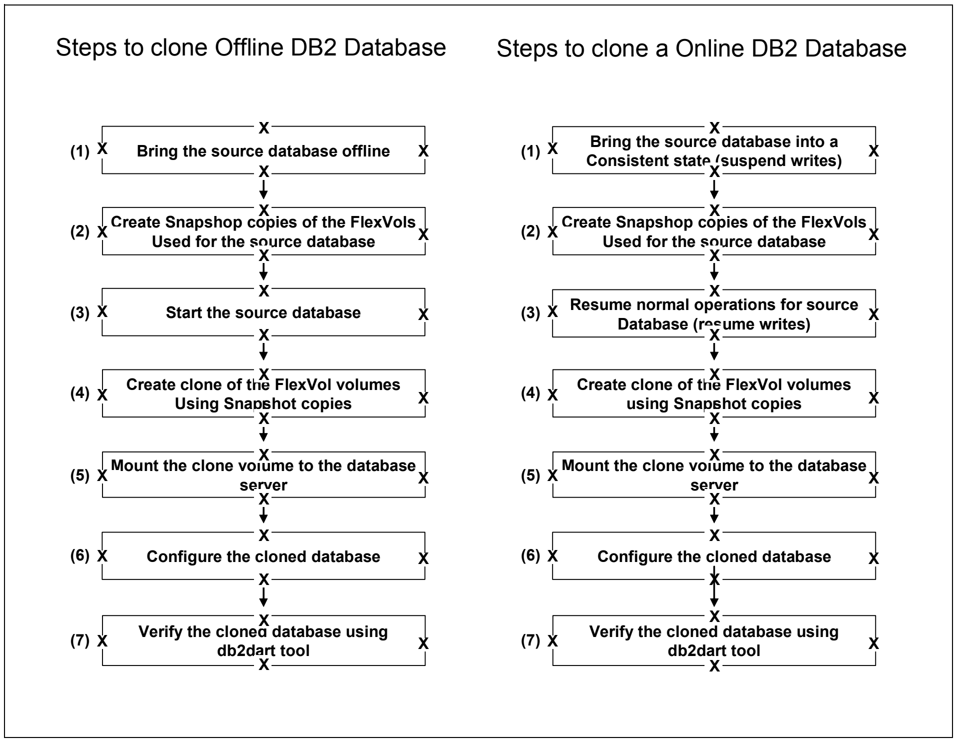


Figure 2-3 Clone database volumes on the second NetApp FAS or IBM N Series storage system

Figure 2-4 Illustrates the steps necessary to create a clone of database in a IBM N series storage system environment.



Steps to clone Offline DB2 Database
(1) Bring the source database offline
(2) Create Snapshop copies of the FlexVols Used for the source database
(3) Start the source database
(4) Create clone of the FlexVol volumes Using Snapshot copies
(5) Mount the clone volume to the database server
(6) Configure the cloned database
(7) Verify the cloned database using db2dart tool
Steps to clone a Online DB2 Database
(1) Bring the source database into a Consistent state (suspend writes)
(2) Create Snapshop copies of the FlexVols Used for the source database
(3) Resume normal operations for source Database (resume writes)
(4) Create clone of the FlexVol volumes using Snapshot copies
(5) Mount the clone volume to the database server
(6) Configure the cloned database
(7) Verify the cloned database using db2dart tool

Figure 2-4 Cloning DB2 database in NetApp FAS or IBM N Series storage system environment

3

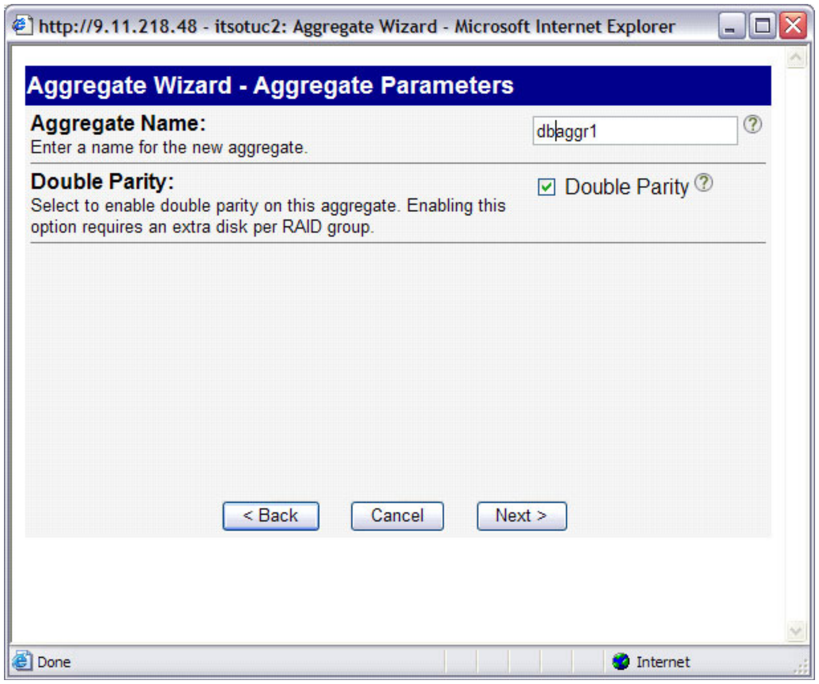
Creating a DB2 UDB Database on an IBM N series

To create a DB2 UDB database on a IBM N-Series storage system, you need to perform a few configuration steps on the database server and the IBM N-Series storage system.

3.1 Create containers

After configuring the database server and the storage system, you need to create the following storage containers and objects on your storage system.

- An aggregate named *dbaggr1*


Aggregate Wizard - Aggregate Parameters

Aggregate Name: dbaggr1
Enter a name for the new aggregate.

Double Parity: ✓ Double Parity
Select to enable double parity on this aggregate. Enabling this option requires an extra disk per RAID group.

http://9.11.218.48 - itsotuc2: Aggregate Wizard - Microsoft Internet Explorer
< Back
Cancel
Next >
Done
Internet

Figure 3-1 dbaggr1

- Flexible volumes named *dbdata* and *dblogs* within aggregate dbaggr

For SAN environments, you need to perform the following additional steps:

- Create a LUN named */vol/dbdata/data* within FlexVol volume dbdata.
- Create a LUN named */vol/dblogs/logs* within FlexVol volume dblogs.
- Create an igroup named *dbhost1_fcp_igp* for the database server dbhost1.
- On the storage system named dbstore1, create mappings for the LUNs named */vol/dbdata/data* and */vol/dbdata/logs* to the igroup named dbhost1_fcp_igp, using ID 0 and 1 respectively.

After you complete these initial steps, the storage system should be ready to receive a DB2 UDB database, which you can create by completing the steps in the following sections.

3.2 Create an instance

If an instance does not already exist, log in as the user root on the database server and create a DB2 instance by executing the following command:

```
[DB2Dir]/instance/db2icrt -u [FencedID] [InstanceName]
```

Where the following conditions are defined:

- DB2Dir identifies the directory where the DB2 UDB software was installed:
  - On AIX®, the DB2 installation directory for version 8.1 is /usr/opt/db2_08_01.
  - On all other UNIX-based operating systems, the installation directory for version 8.1 is /opt/IBM/db2/V8.1.
- FencedID identifies the ID of the user under which fenced user-defined functions and fenced stored procedures will run.
- InstanceName identifies the name that is to be assigned to the new instance.

For example, to create a database instance named db2inst1, you would execute the following command on the database server:

```
/opt/IBM/db2/V8.1/instance/db2icrt -u db2inst1 db2inst1
```

3.3 Create the database

Log in as the database instance owner and execute the following command on the database server:

```
db2 "create database [DatabaseName] on [MountPoint]"
```

Where the following terms are defined:

- DatabaseName identifies the name that is to be assigned to the new database once it has been created.
- MountPoint identifies the mount point location where the new database is to be created.

For example, to create a database named mydb on a storage system volume that is mounted on a mount point named /mnt/dbdata, you would execute the following command on the database server:

```
 db2 "create database mydb on /mnt/dbdata"
```

3.4 Change location of transaction logs

It is not a good practice to leave transaction log files in their default location on the default volume. In fact, when a DB2 UDB database is stored on an IBM N Series storage system volume, the transaction log files for the database should be stored on a separate volume. To change the location where transaction log files are stored, execute the following command at the database server:

```
db2 update db cfg for [DatabaseName] using NEWLOGPATH [NewLogLocation]
```

Where the following terms are defined:

- `DatabaseName` identifies the name assigned to the database whose log file storage location is to be changed.
- `NewLogLocation` identifies the new location where the database's transaction log files are to be stored.

For example, to change the log directory from the default location to a directory named /mnt/dblogs, you would execute the following command on the database server:

```
db2 update db cfg for mydb using NEWLOGPATH /mnt/dblogs
```

Note: The new log path setting will not become effective until all the users are disconnected and the database is deactivated. When first connection is made after reactivating the database, the database manager will move the transaction log files to the new location.

3.5 Switch from circular logging to archive logging

By default, DB2 UDB uses a circular logging mechanism for the database, and the transaction logs are stored in a subdirectory where the database was created. However, most production databases run in archive logging mode to support roll-forward recovery. Switching a DB2 UDB V8 database from circular logging to archive logging is easy, and it can be done by updating the primary log archive method configuration parameter (named LOGARCHMETH1).

You can update the LOGARCHMETH1 configuration parameter by executing the following command on the database server:

```
db2 update db cfg for [DatabaseName] using LOGARCHMETH1
DISK:[ArchiveDir]
```

In this command, the following terms are defined:

- `DatabaseName` identifies the name assigned to the database whose logging method is to be changed.
- `ArchiveDir` identifies the directory (location) where archived transaction log files are to be stored.

For example, to enable archive logging for the DB2 database named mydb and place the archive file in a directory named /mnt/dbarch1/mydb, you would execute the following command on the database server:

```
db2 update db cfg for mydb using LOGARCHMETH1 DISK:/mnt/dbarch1/mydb
```

If you want to retain a second copy of archive log files on another disk, you need to update the secondary log archive method configuration parameter, named LOGARCHMETH2. You can update the LOGARCHMETH2 configuration parameter by executing the following command on the database server:

```
db2 update db cfg for [DatabaseName] using LOGARCHMETH2
DISK:[ArchiveDir]
```

Where the following terms are defined:

- `DatabaseName` identifies the name assigned to the database for which duplex logging is to be enabled.
- `ArchiveDir` identifies the directory (location) where the second copy of the archived transaction log files is to be stored.

For example, to store and maintain a second set of archive logs for a DB2 database named `mydb` in a directory named `/mnt/dbarch2/mydb`, you would execute the following command on the database server:

```
db2 update db cfg for mydb using LOGARCHMETH2 DISK:/mnt/dbarch2/mydb
```

3.6 Create offline copy

After the logging method is changed from circular to archival, the database is placed into Backup Pending state and can no longer be used until a full offline backup copy of the database has been created. You can create an offline backup copy of the database by executing the following commands on the database server:

```
db2 force application all
db2 backup database [DatabaseName] to [BackupDir]
```

In these command, the following terms are defined:

- `DatabaseName` identifies the name assigned to the database that is to be backed up.
- `BackupDir` identifies the directory (location) where backup images are to be stored.

For example, to create an offline backup copy of the database named mydb, you would execute the following command on the database server:

```
db2 force application all
db2 backup database mydb to /dbbackup
```

After completing these steps, the production database is ready. You can create database objects and start using the database.

4

Cloning a DB2 UDB database in the N series environment

This chapter discusses cloning databases in DB2 using IBM N series systems.

4.1 Select a database server to access the cloned database

You need to select a database server that will be used to access the cloned database. You have the following choices:

4.1.1 Select the production database server and database

Select the database server which accesses the production (parent) database. In this case, you can uses existing DB2 instance or create a new one using the same DB2 UDB V8 code as the production instance. In order to create new DB2 instance you need to complete the following steps:

1. Switch the authority to the user root and create a user named db2instc on the database server by executing the following command:

   ```
   useradd -c "DB2 clone db instance owner" -u 710 -g db2adm -G db2adm
   db2instc -p db2instc
   ```

The new user will own the DB2 instance used to access the cloned database.

2. Now create a new DB2 instance using the same DB2 code as the production database instance. In order to do that you need execute the following command on the database server:

   ```
   [DB2InstallationPath]/instance/db2icrt -u [FencedUser]
   [InstanceName]
   ```

   Where the following terms are defined:

   – `DB2InstallationPath` identifies the directory where the DB2 UDB V8 code was installed.

   – `FencedUser` identifies the ID of the user under which fenced user-defined functions and fenced stored procedures will run.

   – `InstanceName` identifies the name that is to be assigned to the new instance.

   For example, if the DB2 UDB V8 were installed in /opt/IBM/db2/V8.1 directory, you would execute the following command on the database server to create a DB2 instance named db2instc:

   ```
   /opt/IBM/db2/V8.FP11/instance/db2icrt -u db2instc db2instc
   ```

   a. Check the list of instances and DB2 UDB code used by executing the following command on the database server.

      ```
      /opt/IBM/db2/V8.FP11/instance/db2ilist -a
      ```

The output from the command should look similar to Example 4-1.

Example 4-1 The db2ilist output

```
db2inst1   32   /opt/IBM/db2/V8.1
db2instc   32   /opt/IBM/db2/V8.1
```

4.1.2 Select a database server that has a different DB2 UDB version

A database server other than the production database server, which has a different version of DB2 UDB from that of the production DB2 UDB Version. In this case, you need to install the same DB2 UDB V8 code as the production database on the database server and create a database instance as described in the 4.1.1, "Select the production database server and database" on page 26. The new instance name can be same as the production DB2 instance provided there is no other instance with the same name on this server.

4.1.3 Select a non-production server without DB2 UDB installed

A non-production server is a database server other than the production database server that does not have DB2 UDB installed.

In this case, you must install the same DB2 UDB V8 code as the production database on the database server and create a database instance as described in 4.1.1, "Select the production database server and database" on page 26. The new instance name can be same as the production DB2 instance.

4.2 Clone an offline database on the same storage system

To clone a database that is offline, complete the steps in the following sections.

4.2.1 Bring the source database offline

To bring the database offline, terminate all the application connections to the database that is to be cloned by executing the following command on the database server:

```
db2 force applications all
```

For example, to terminate all application connections, execute the following command on the database server:

```
db2 force applications all
```

4.2.2 Create Snapshot copies of the database FlexVol volumes

Create a Snapshot copy of each FlexVol volume that is used for the production database by executing the following command on the storage system:

```
snap create [VolName] [SnapName]
```

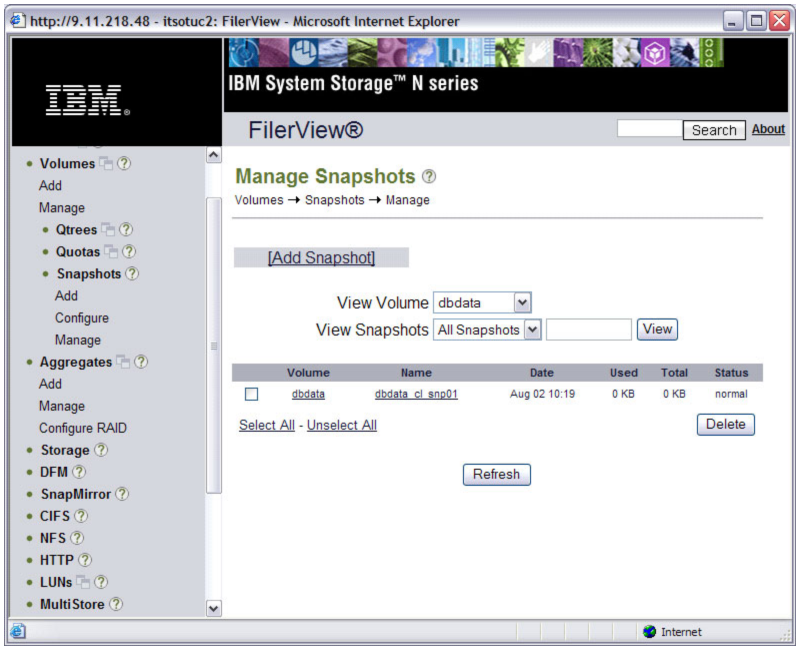
Where the following terms are defined:

- `VolName` identifies the name assigned to the FlexClone volume that is to be created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named dbdata_snap01 for a FlexVol volume named dbdata, execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp01
```

The result is shown in Figure 4-1.



Volume	Name	Date	Used	Total	Status
dbdata	dbdata_cl_snp01	Aug 02 10:19	0 KB	0 KB	normal

Figure 4-1 Snapshot results

It is recommended that you develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purposes.

4.2.3 Start the source database

When the Snapshot copies of the FlexVol volumes of the source database are created, you can connect to the source database and start using it.

4.2.4 Clone the FlexVol volumes

Create a clone of each FlexVol volume using the Snapshot copies created on step 4.2.2, “Create Snapshot copies of the database FlexVol volumes” on page 28. Create a clone volume by executing this command on the storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined:

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume snapshot that is used as source for the clone volume.

For example, to create a clone volume of a FlexVol volume named dblogs using the Snapshot copy named dbdata_cl_snp.01, execute the following command on the N series storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.01
```

Verify the clone you just created with the following command. You can see the results in Example 4-2.

```
vol status dbdata_cl
```

Example 4-2 The vol status command

```
itsotuc2> vol status dbdata_cl
         Volume State      Status            Options
      dbdata_cl online     raid_dp, flex     create_ucode=on,
                                             convert_ucode=on
                Clone, backed by volume 'dbdata', snapshot
'dbdata_cl_snp.01'
                Containing aggregate: 'aggr0'
```

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

Note: A Snapshot copy is not required to create a clone of a FlexVol volume. If you do not explicitly create a Snapshot copy and specify it when executing the **`vol clone`** command, a Snapshot copy will be implicitly created and used for the clone volume. A Snapshot copy created implicitly will have a system assigned name. We recommend explicitly creating a Snapshot and assigning it a meaningful name before creating a clone FlexVol volume.

4.2.5 Create an export entry for the clone volume

To mount a clone volume to the database server, you must create an export entry for it in the /etc/exports file that resides on the IBM N Series storage system. The export entry can be created by executing the following command on the IBM N Series storage system:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

Where the following variables are defined:

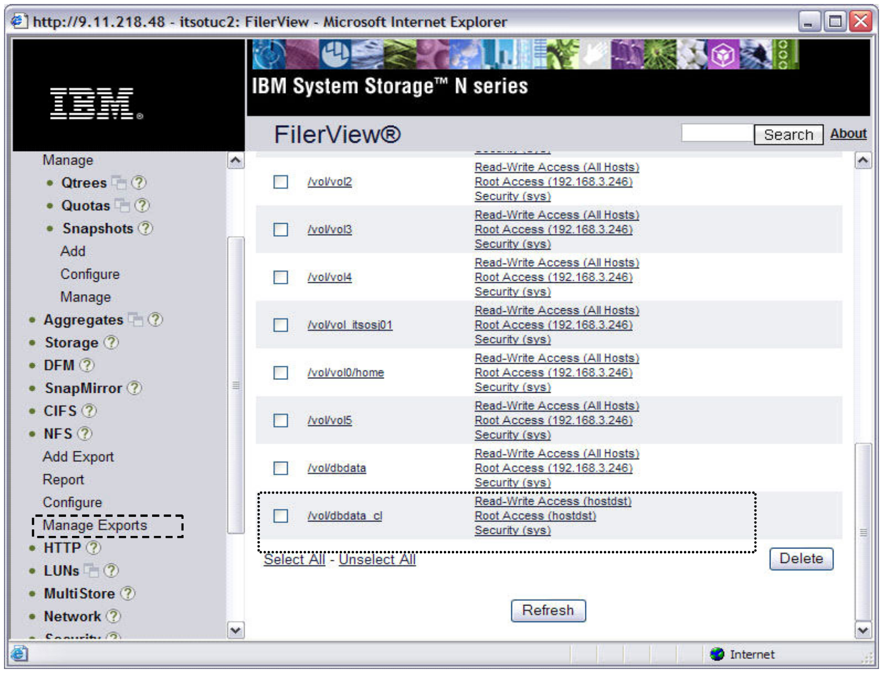
- `HostName` identifies the name assigned to the database server.
- `PathName` identifies the name assigned to the flexible volume.

For example, to create an export entry for a clone volume named dbdata_cl and allow root access from the database server named *hostdst* for it, execute the following command on the storage system:

```
exportfs -p rw=hostdst,root=hostdst /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is used for the clone database.

You can verify your exports with FilerView® in Figure 4-2.



☐	/vol/vol2	Read-Write Access (All Hosts) Root Access (192.168.3.246) Security (sys)
☐	/vol/vol3	Read-Write Access (All Hosts) Root Access (192.168.3.246) Security (sys)
☐	/vol/vol4	Read-Write Access (All Hosts) Root Access (192.168.3.246) Security (sys)
☐	/vol/vol_itsosj01	Read-Write Access (All Hosts) Root Access (192.168.3.246) Security (sys)
☐	/vol/vol0/home	Read-Write Access (All Hosts) Root Access (192.168.3.246) Security (sys)
☐	/vol/vol5	Read-Write Access (All Hosts) Root Access (192.168.3.246) Security (sys)
☐	/vol/dbdata	Read-Write Access (All Hosts) Root Access (192.168.3.246) Security (sys)
☐	/vol/dbdata_cl	Read-Write Access (hostdst) Root Access (hostdst) Security (sys)

Select All - Unselect All

Delete

Refresh

Figure 4-2 Verifying the exports

4.2.6 Mount the clone volumes

The clone database can be accessed from the same database server that is used to access the source database, or from a completely different server. The scenarios described in this paper were produced using a second database sever to access the clone database.

To access the clone database, you must mount the clone volumes to a database server. First, you must create a mount point for each clone volume and append a mount entry to the /etc/fstab file. The mount entry should specify the mount options, and it should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

Where the following variables are defined as:

- `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- `FlexVolName` identifies the name assigned to the clone volume.

- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named dbdata_cl that resides on a storage system named srcstore, you would append the following entry to the /etc/fstab file on the database sever:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

Where `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has mount entry specified in the /etc/fstab file, you would execute the following command on the second database server named hostdst:

```
mount /mnt/dbdata_cl
```

The database servers we used had a Linux operating system.

In order to operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where the following variables are defined as:

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the files system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named /mnt/dbdata_cl, execute the following command on the second database server:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

4.2.7 Configuring the cloned database

The clone volumes created in 4.2.4, "Clone the FlexVol volumes" on page 29 and mounted in 4.2.6, "Mount the clone volumes" on page 31 are going to be used as the storage containers for the cloned database. You can skip the parts 1 on page 40 and 2 on page 41 of this step if the following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is the same as the production or source database instance name.
- The mount points that are used to mount the clone volumes have the same name as the mount points that are used to mount volumes of the production database.

1. By default when the database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   ```
   [DBDir]/[InstanceName]/NODE000n
   ```

   Where the following terms are defined:

   – `DBDir` identifies the name assigned to the directory or device upon which the database is created.

   – `InstanceName` identifies the name assigned to the DB2 instance to which the database belongs.

     For example, if the DB2 instance name is db2inst1 and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory:

     ```
     /mnt/dbdata/db2inst1/NODE0000
     ```

   For a database server, the DB2 instance name must be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, you must change the default tablespace container's path name by executing following command on the database server:

   ```
   mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n
   ```

   Where the following variables are defined:

   – `DBDir` identifies the name assigned to the directory or device on which the database is created.

   – `OldInstanceName` identifies the name assigned to the DB2 instance to which the production database belongs.

   – `NewInstanceName` identifies the name assigned to the DB2 instance to which the clone database belongs.

For example, to access the clone database from the instance named db2instc, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

2. Change the database name and tablespace containers header information using the **db2inidb** command as specified in 1.5, “The db2inidb command” on page 6. To do this, you must create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to Example 4-3.

Example 4-3 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is mydb and it has its logs on /mnt/dblogs and data on /mnt/dbdata . The clone database is to be renamed to *mydbcl*; it has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is db2inst1 and clone database instance name is db2instc.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing **db2start**. The production database was offline during the Snapshot creation process, therefore, you can connect to the clone database and start using it.

4.2.8 Catalog the source database if necessary

If the cloned database is accessed from same DB2 instance as the production database on the production database server, then on execution the **db2relocatedb** command uncatalogs the source database. Therefore, you must catalog the source database by executing the following command on the database server:

```
db2 "catalog database [DatabaseName] as [DatabaseAlias] on [FileSystem]"
```

Where the following terms are defined:

- `DatabaseName` identifies the name assigned to the database that is being cataloged.
- `DatabaseAlias` identifies the alias name assigned to the database that is being cataloged.
- `FileSystem` specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named `mydb` that resides on file system named `/mnt/dbdata`, execute the following command on the database server:

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

4.2.9 Verify the database

After performing the previous steps, check the entire clone database for architectural correctness. You can do that by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

Where `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to test the cloned database named *mydbcl*, execute the following command on the database server:

```
db2dart mydbcl /db
```

The **db2dart** utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the <$HOME>/sqllib/db2dump/DART0000/ directory. There is a summary at the end of report. Read the summary and check to see if there are any errors.

The additional information about accessing a cloned database from the same database server as the source is described in Appendix A, “Configuring UNIX to access cloned and source databases in an NAS environment” on page 99.

4.3 Clone an online database on the same storage system

After reading 4.2, “Clone an offline database on the same storage system” on page 27 you should have a fair understanding of cloning an offline DB2 database. In this section, you learn how to clone an online DB2 database by completing the following steps.

4.3.1 Bring the database into a consistent state (suspend writes)

In this scenario, the source database is online. Therefore, to prevent partial page writes while database cloning is in progress, the database write operations must be temporarily suspended by executing the following command on the database server:

```
db2 set write suspend for database
```

The **`set write suspend for database`** command causes the DB2 database manager to suspend all write operations to tablespace containers and log files that are associated with the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The FlexVol volume clone process is completed very quickly, so the database does not need to stay in write suspend mode for more than a few seconds.

4.3.2 Create Snapshot copies of the database FlexVol volumes

Next, create a Snapshot copy of each FlexVol volume that is used for the production database.

A Snapshot copy of a FlexVol volume can be created by executing the following command on the storage system:

```
snap create [VolName] [SnapName]
```

Where the following variables are defined:

- `VolName` identifies the name assigned to the FlexClone volume that is to be created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_snap.1` for a FlexVol volume named `dbdata`, execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.1
```

It is recommended that you develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purpose.

4.3.3 Resume normal database operations (resume writes)

After Snapshot copies are created, you must resume write operations to the database by executing the following command on the database server:

```
set write resume for database
```

4.3.4 Clone the FlexVol volumes.

Create a clone of each FlexVol volume using the Snapshot copies created in 4.3.2, “Create Snapshot copies of the database FlexVol volumes” on page 36. A clone volume can be created by executing the following command on the storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined:

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume snapshot that is used as source for the clone volume.

For example, to create a clone volume of a FlexVol volume named `dblogs` using the Snapshot copy named `dbdata_cl_snp.01`, execute the following command on the N series storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.01
```

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

Note: A Snapshot copy is not required to create a clone of a FlexVol volume. If you do not explicitly create a Snapshot copy and specify it when executing the **`vol clone`** command, a Snapshot copy will be implicitly created and used for the clone volume. A Snapshot copy created implicitly will have a system assigned name. We authors recommend explicitly creating a Snapshot and assigning it a meaningful name before creating a clone FlexVol volume.

4.3.5 Create NFS export entries for the cloned volumes

In order to mount a clone volume to the database server, you must create an export entry for it in the /etc/exports file that resides on the storage system. The export entry can be created by executing the following command on the storage system:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

Where the following variables are defined as:

- `HostName` identifies the database server name that will use these clone volumes as storage.
- `PathName` identifies the clone volume path.

For example, to create an export entry for a clone volume named dbdata_cl and allow root access to it from a database server named dbhost1, you would execute the following command on the storage system:

```
exportfs -p rw=dbhost1,root=dbhost1 /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is to be used for the database in the test environment.

4.3.6 Mount the cloned volumes

To access the clone database, you must mount the clone volume to a database server. First, you create a mount point for each clone volume and append a mount entry to the /etc/fstab file. The mount entry should specify the mount options, and it should look similar to Example 4-4.

Example 4-4 The /etc/fstab file

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

In this example, the following terms are defined:

- `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- `FlexVolName` identifies the name assigned to the clone volume.
- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named *dbdata_cl* that resides on a storage system named *srcstore*, append the following entry to the /etc/fstab file on the database sever:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

Where `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume with the mount entry specified in the /etc/fstab file, execute the following command on the second database server, named hostdst:

```
mount /mnt/dbdata_cl
```

The database servers we used had Linux operating systems.

To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where the following variables are defined as:

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the files system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, you would execute the following command on the second database server:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

4.3.7 Configuring the cloned database

You use the clone volumes that you created in 4.3.4, “Clone the FlexVol volumes.” on page 37 and mount in 4.3.6, “Mount the cloned volumes” on page 38 as the storage containers for the cloned database. You can skip the steps 1 and 2 if following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is same as the production or source database instance name.
- The mount points that are used to mount the clone volumes have same name as the mount points that are used to mount volumes of the production database.

To configure the cloned database, follow these steps:

1. By default when database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   ```
   [DBDir]/[InstanceName]/NODE000n
   ```

   Where the following variables are defined as:

   – `DBDir` identifies the name assigned to the directory or device on which the database is created.

   – `InstanceName` identifies the name assigned to the DB2 instance to which the database belongs.

   For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory.

   ```
   /mnt/dbdata/db2inst1/NODE0000
   ```

   For a database server, the DB2 instance name must be unique. Therefore, you must create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, you must change the default tablespace container's path name by executing following command on the database server:

   ```
   mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n
   ```

Where the following variables are defined as:

- `DBDir` identifies the name assigned to the directory or device on which the database is created.
- `OldInstanceName` identifies the name assigned to the DB2 instance to which the production database belongs.
- `NewInstanceName` identifies the name assigned to the DB2 instance to which the clone database belongs.

For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

2. Change the database name and tablespace containers header information using the **`db2inidb`** command as specified in 1.5, "The db2inidb command" on page 6. To do this, you must create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario looks similar to that shown in Example 4-5.

Example 4-5 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is mydb and it has its logs on /mnt/dblogs and data on /mnt/dbdata . The clone database is to be renamed to *mydbcl*; it has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is db2inst1 and clone database instance name is db2instc.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing **`db2start`**.

If the production database was online during the Snapshot creation process, you must use the **db2inidb** utility to initialize the clone database as a snapshot. You can do this by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

Where `DatabaseName` identifies the name of the clone database that is going to be used for the test environment. For example, to initialize a clone database named *mydbcl*, execute the following commands on the database server:

```
db2inidb mydbcl as snapshot
```

4.3.8 Verify the database

After performing the previous steps, you must check the entire database for architectural correctness. You can do so by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

Where `DatabaseName` identifies the name of the clone database used for the test environment. For example, to test the database named *mydbcl*, execute the following command on the database server:

```
db2dart [mydbcl] /db
```

4.4 Clone an offline database to a remote storage system

The FlexClone feature combined with SnapMirror (Figure 4-3 on page 43) makes it possible to clone a DB2 database on a remote IBM N Series storage system over the network or on another aggregate of the same IBM N Series storage system. This section describes the steps necessary to clone a database on the second storage system at a remote location.

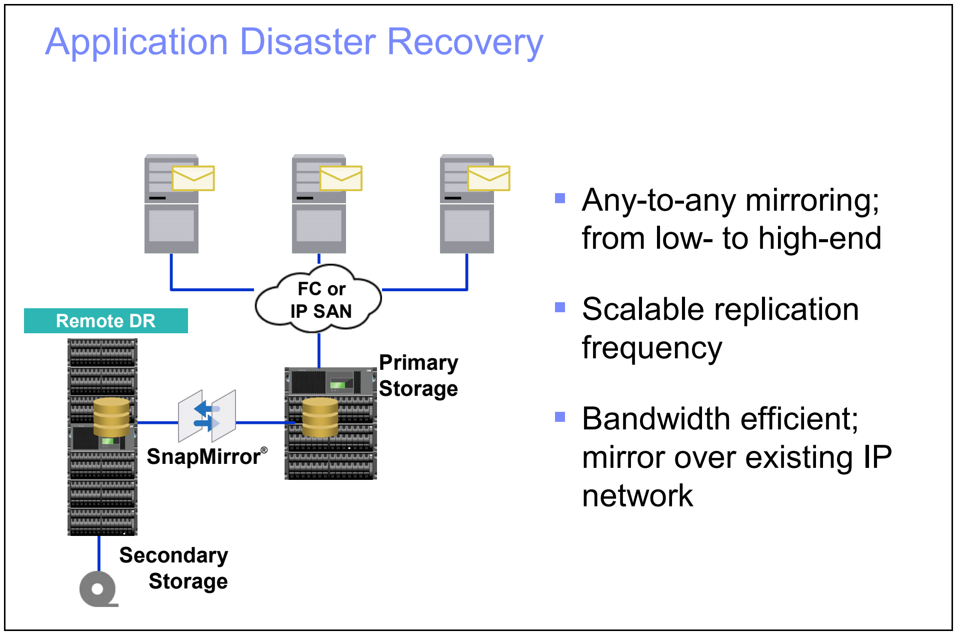

Application Disaster Recovery
FC or IP SAN
Remote DR
Primary Storage
SnapMirror®
Secondary Storage
Any-to-any mirroring; from low- to high-end
Scalable replication frequency
Bandwidth efficient; mirror over existing IP network

Figure 4-3 SnapMirror utilization

4.4.1 Configure SnapMirror

To configure SnapMirror, complete the following steps on the SnapMirror source and the SnapMirror destination IBM N Series storage system:

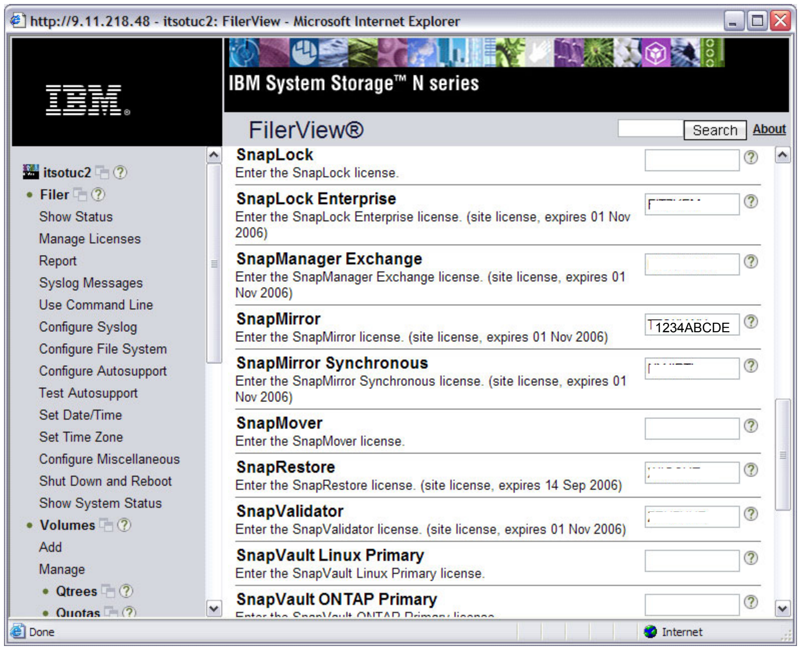
1. Add licenses for SnapMirror and SnapMirror sync on the source and destination IBM N Series storage system by executing the following command:

   ```
   license add  [licenseCode]
   ```

   Where `licenseCode` identifies the license key for the product or feature that is provided to you by the NetApp or IBM sales representative. For example, to enable a SnapMirror license key, execute the following command on the storage system:

   ```
   License add  1234ABCDE
   ```

Verify the license through FilerView, as shown in Figure 4-4.


http://9.11.218.48 - itsotuc2: FilerView - Microsoft Internet Explorer

IBM System Storage™ N series

FilerView®

Search | About

- itsotuc2
- Filer
  - Show Status
  - Manage Licenses
  - Report
  - Syslog Messages
  - Use Command Line
  - Configure Syslog
  - Configure File System
  - Configure Autosupport
  - Test Autosupport
  - Set Date/Time
  - Set Time Zone
  - Configure Miscellaneous
  - Shut Down and Reboot
  - Show System Status
- Volumes
  - Add
  - Manage
  - Qtrees
  - Quotas

SnapLock
Enter the SnapLock license.

SnapLock Enterprise
Enter the SnapLock Enterprise license. (site license, expires 01 Nov 2006)

SnapManager Exchange
Enter the SnapManager Exchange license. (site license, expires 01 Nov 2006)

SnapMirror
Enter the SnapMirror license. (site license, expires 01 Nov 2006) 1234ABCDE

SnapMirror Synchronous
Enter the SnapMirror Synchronous license. (site license, expires 01 Nov 2006)

SnapMover
Enter the SnapMover license.

SnapRestore
Enter the SnapRestore license. (site license, expires 14 Sep 2006)

SnapValidator
Enter the SnapValidator license. (site license, expires 01 Nov 2006)

SnapVault Linux Primary
Enter the SnapVault Linux Primary license.

SnapVault ONTAP Primary

Done | Internet

Figure 4-4 License manager

2. Enable the SnapMirror feature by executing the following command on the SnapMirror source and destination storage systems:

   ```
   snapmirror on
   ```

   Alternatively, you can use this command:

   ```
   options snapmirror.enable on
   ```

   Verify options with the following command:

   ```
   options snapmirror
   ```

   The output looks similar to that shown in Example 4-6.

Example 4-6 The options snapmirror command

```
itsotuc2> options snapmirror
snapmirror.access              host=192.168.2.2
snapmirror.checkip.enable      off
snapmirror.delayed_acks.enable on
snapmirror.enable              on
snapmirror.log.enable          on
```

3. On the SnapMirror source storage system, create a snapmirror.allow file that resides in the /vol/vol0/etc directory and append an entry for the SnapMirror destination storage system. The entries in the snapmirror.allow file should look similar to the following:

   ```
   [DestinationStorageSystemName]
   ```

   Where `DestinationStorageSystemName` identifies the name assigned to the SnapMirror destination storage system.

   For example, to append an entry for the storage system named *dststore*, add the following entry to the snapmirror.allow file on the SnapMirror source storage system:

   ```
   dststore
   ```

4. Create a configuration file named snapmirror.conf that resides in the /volvol0/etc directory on the destination storage system. The configuration file is used to identify the source storage system, source volume, destination volume, and SnapMirror schedule. A snippet from a snapmirror.conf file looks similar to that shown in Example 4-7.

Example 4-7 A snippet from a snapmirror.conf file

```
#----------------------------------------------------------------------------
# file name: snapmirror.conf
# Description: This file resides on a destination storage system and
# consists of SnapMirror configuration details. Entry format is as follows:
# src_filer:[vol | qtreepath] dest_filer:[vol | qtreepath] argument schedule
#----------------------------------------------------------------------------
srcstore:dbdata dststore:dbdata - 0 * * 1-5
srcstore:dblogs dststore:dblogs - 0 * * 1-5
# srcstore:dblog1 dststore:dblog1 - sync   -- synchronous SnapMirror
```

If the word *sync* is specified in the snapmirror.conf file instead of a definitive schedule, that implies synchronous Snapmirror configuration for that particular volume.

4.4.2 Initialize SnapMirror

As noted earlier, the initialization process transfers data, including all Snapshot copies, from the source volume to the destination volume for the first time. Thereafter, only changed blocks are transferred. In order to initialize the SnapMirror relationship, you must restrict the destination volume by executing the following command on the destination storage system:

```
vol restrict [VolumeName]
```

Where VolumeName identifies the name assigned to the volume that is being used as the SnapMirror destination. For example, to restrict a SnapMirror destination volume named dbdata, you would execute the following command on the destination storage system:

```
vol restrict dbdata
```

Verify the execution of the command with the following command;

```
vol status dbdata
```

Example 4-8 shows the output from this command.

Example 4-8 Output from the vol status command

```
vol status dbdata
Volume State      Status            Options
dbdata restricted raid_dp, flex
       Volume has clones: dbdata_cl
       Containing aggregate: 'aggr0'
```

After restricting the destination FlexVol volumes, you must initialize the SnapMirror relationship for each volume that is used for the database by executing the following command on the destination storage system:

```
snapmirror initialize -S
[SourceStorageSystem]:[VolumeName][DestinationStorageSystem]:[VolumeName]
```

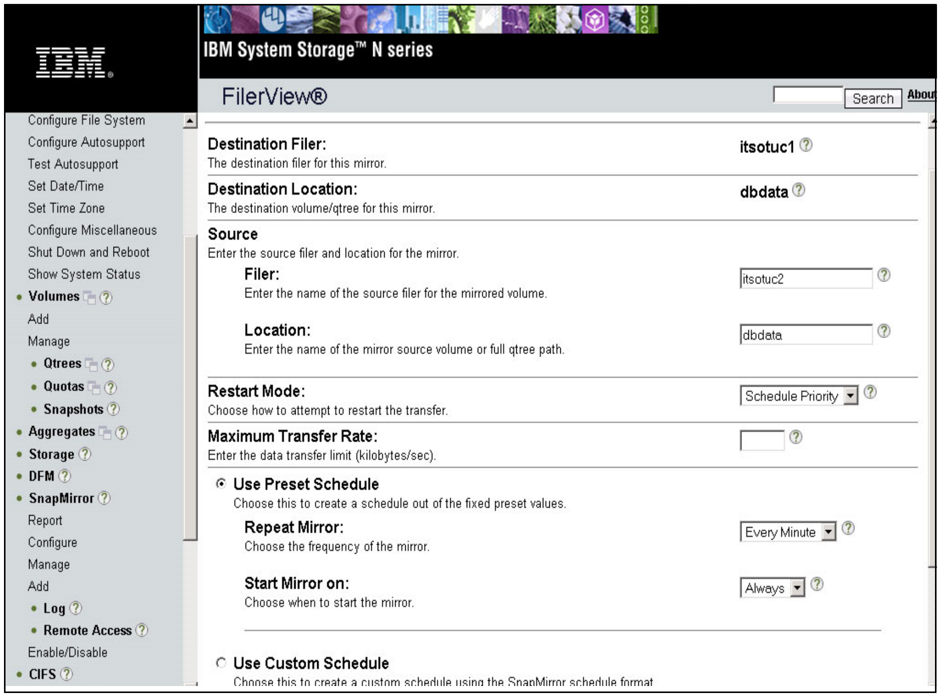
Where the following variables are defined as:

- SourceStorageSystem identifies the name assigned to the SnapMirror source storage system.
- VolumeName identifies the name assigned to the volume that is the SnapMirror source/destination.
- DestinationStorageSystem identifies the name assigned to the SnapMirror destination storage system.

For example, to initialize the SnapMirror relationship that is specified in the configuration file for the volume named dbdata, execute the following command on the destination storage system:

```
snapmirror initialize -S srcstore:dbdata dststore:dbdata
```

You can accomplish the same thing through FilerView (Figure 4-5).



Destination Filer: itsotuc1
The destination filer for this mirror.

Destination Location: dbdata
The destination volume/qtree for this mirror.

Source
Enter the source filer and location for the mirror.

Filer: itsotuc2
Enter the name of the source filer for the mirrored volume.

Location: dbdata
Enter the name of the mirror source volume or full qtree path.

Restart Mode: Schedule Priority
Choose how to attempt to restart the transfer.

Maximum Transfer Rate:
Enter the data transfer limit (kilobytes/sec).

[✓ **Use Preset Schedule**]
Choose this to create a schedule out of the fixed preset values.

Repeat Mirror: Every Minute
Choose the frequency of the mirror.

Start Mirror on: Always
Choose when to start the mirror.

Use Custom Schedule
Choose this to create a custom schedule using the SnapMirror schedule format

Figure 4-5 SnapMirror creation

The SnapMirror initialize command creates a Snapshot copy of the source volume and transfers data from the source to the destination volume. After baseline data transfer is finished, the destination volume data is available in read-only mode.

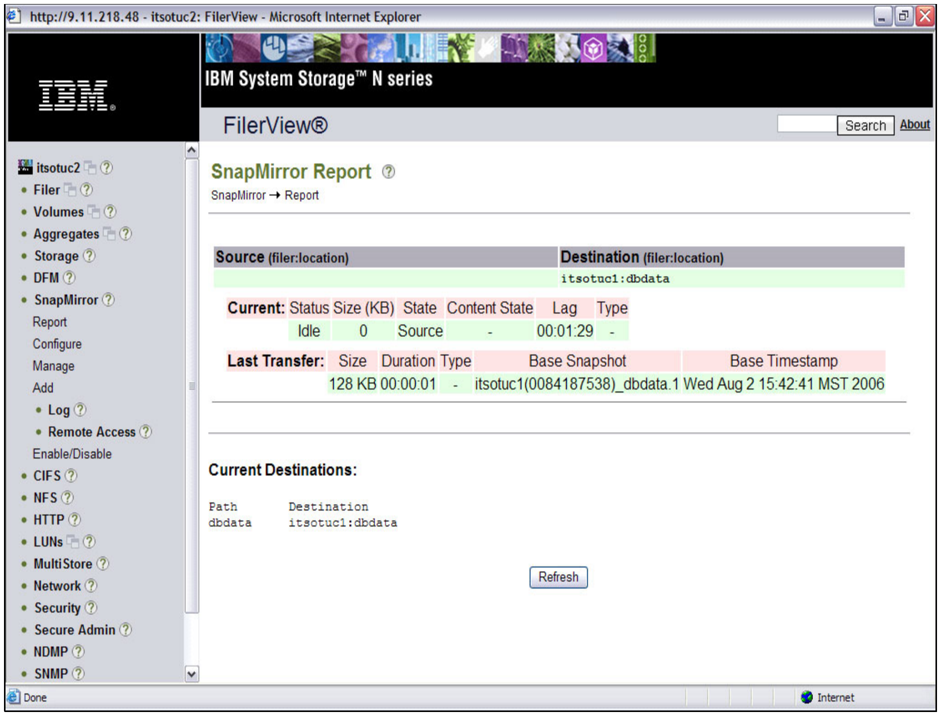
The progress of SnapMirror can be checked by executing the following command on the storage system:

```
snapmirror status
```

The output from this command should look similar to that shown in Example 4-9 or Figure 4-6.

Example 4-9 Output from the snapmirror status command

```
snapmirror status
snapmirror is on.
Source          Destination      State          Lag      Status
srcstore:dbdata dststore:dbdata  SnapMirrored   00:07:33 Idle
srcstore:dblogs dststore:dblogs  Uninitialized  - Transferring(116 MB
done)
```



SnapMirror Report

SnapMirror → Report

Source (filer:location)	Destination (filer:location)
	itsotuc1:dbdata

Current:	Status	Size (KB)	State	Content State	Lag	Type
	Idle	0	Source	-	00:01:29	-

Last Transfer:	Size	Duration	Type	Base Snapshot	Base Timestamp
	128 KB	00:00:01	-	itsotuc1(0084187538)_dbdata.1	Wed Aug 2 15:42:41 MST 2006

Current Destinations:

```
Path      Destination
dbdata    itsotuc1:dbdata
```

Figure 4-6 SnapMirror Status

4.4.3 Create clones of the FlexVol volumes

Create clones of the FlexVol volumes on the remote storage system.In the SnapMirror relationship, the destination volumes are read-only; therefore, to create a clone database on a remote IBM N series storage system, you need to create Snapshot copies of the FlexVol volumes on the SnapMirror source storage system and update the SnapMirror relationship manually. The SnapMirror update operation replicates data, including Snapshot copies, to the

SnapMirror destination storage system, and the replicated Snapshot copy can be used to create the clone volume. Here are the detailed steps:

1. Bring the database offline by terminating all the application connections to the database by executing the following command on the database server:

   ```
   db2 force applications all
   ```

2. Create a Snapshot copy for each volume that is used for the database by executing the following command on the SnapMirror source storage system:

   ```
   snap create [VolumeName] [SnapName]
   ```

   Where the following variables are defined as:

   – `VolumeName` identifies the name assigned to the FlexVol volume on the SnapMirror storage system that is used for the database.

   – `SnapName` identifies the name assigned to the Snapshot copy of the FlexVol volume that is used as the source for the clone volume.

   For example, to create a Snapshot copy named `dbdata_cl_snap.1` of a volume named dbdata, execute the following command on the SnapMirror source storage system:

   ```
   snap create dbdata dbdata_cl_snap.1
   ```

   You should name the Snapshot copy in a way that it describes its intended use. Naming the Snapshot copy appropriately helps to avoid unintended deletion of it from the source storage system.

3. After Snapshot copies of the FlexVol volumes of the source database are created, connect to the database and start using it.

4. Replicate the Snapshot copy created in step 2 on page 49 to the SnapMirror destination volume. In order to do so, you must update the SnapMirror relationship manually by executing the following command on the destination storage system:

   ```
   snapmirror <-S [SourceStorageSystem]:[VolumeName]> update
   [DestinationStorageSystem]:[VolumeName]
   ```

   Where the following variables are defined as:

   – `SourceStorageSystem` identifies the name assigned to the SnapMirror source storage system.

   – `DestinationStorageSystem` identifies the name assigned to the SnapMirror destination storage system.

   – `VolumeName` identifies the name assigned to the source or destination volume in the SnapMirror relationship.

For example, to update SnapMirror for a volume named `dbdata`, execute the following command on the SnapMirror destination storage system named *dststore*:

```
snapmirror -S srcstore:dbdata update dststore:dbdata
```

When the update command is executed for asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including Snapshot copies.

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes visible on the destination volume immediately. Therefore, manual update is not required.

Important: Snapshot copies are also created automatically for SnapMirror update purpose. It is recommended not to use these automatically created Snapshot copies to create a clone volume.

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the source storage system and replicated to the destination. You can create a clone volume from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined as:

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is the source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone volume named dbdata_cl from the Snapshot copy named dbdata_cl_snp.1 of the volume named dbdata, execute the following command on the destination storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.1
```

Important: The Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship will fail.

4.4.4 Create NFS export entries for the cloned volumes

To mount a clone volume to the database server, you must create an export entry for it in the /etc/exports file that resides on the NetApp FAS or IBM N Series storage system. The export entry can be created by executing the following command on the IBM N series storage system:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

Where the following variables are defined as:

- `HostName` identifies the name assigned to the database server.
- `PathName` identifies the name assigned to the flexible volume.

For example, to create an export entry for a clone volume named `dbdata_cl` and allow root access from the database server named hostdst for it, execute the following command on the storage system:

```
exportfs -p rw=hostdst,root=hostdst /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is used for the clone database.

4.4.5 Mount the clone volumes

To access the clone database, you must mount it to a database server.

1. Create a mount point for each clone volume and append a mount entry to the /etc/fstab file. The mount entry should specify the mount options and it should look similar to the following:

   ```
   [StorageSystemName]:[FlexVolName] [MountPoint] nfs
   hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
   ```

   Where the following variables are defined as:

   – `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.

   – `FlexVolName` identifies the name assigned to the clone volume.

   – `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

   For example, for a clone volume named `dbdata_cl` that resides on the storage system named *srcstore*, append the following entry to the /etc/fstab file on the database sever:

   ```
   srcstore:dbdata_cl /mnt/dbdata_cl nfs
   hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
   ```

2. After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

   ```
   mount [MountPoint]
   ```

   Where `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

   For example, to mount a clone volume that has mount entry specified in the /etc/fstab file, execute the following command on the second database server named `hostdst`:

   ```
   mount /mnt/dbdata_cl
   ```

   The database servers we used had Linux operating systems.

3. In order to operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volumes that are mounted on the database server. Ownership can be changed by executing the following command on the database server:

   ```
   chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
   ```

   Where the following variables are defined as:

   – `InstanceOwner` identifies the name assigned to the user who owns the database instance.

   – `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.

   – `FileSystem` identifies the name of the files system whose ownership is to be changed.

   For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, execute the following command on the database server named `hostdst`:

   ```
   chown -R db2inst1:db2adm /mnt/dbdata_cl
   ```

4.4.6 Configure the cloned database

You use the clone volumes that you created in 4.4.3, “Create clones of the FlexVol volumes” on page 48 and mount in the 4.4.5, “Mount the clone volumes” on page 51 in this section as the storage containers for the cloned database. You can skip steps 1 and 2 if following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is same as the production or source database instance name.
- The mount points that are used to mount the clone volumes have same name as the mount points that are used to mount volumes of the production database.

To configure the cloned database, follow these steps:

1. By default when database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   ```
   [DBDir]/[InstanceName]/NODE000n
   ```

   Where the following variables are defined as:

   – `DBDir` identifies the name assigned to the directory or device on which the database is created.

   – `InstanceName` identifies the name assigned to the DB2 instance to which the database belongs.

   For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace containers will reside in the following directory.

   ```
   /mnt/dbdata/db2inst1/NODE0000
   ```

   For a database server the DB2 instance name has to be unique. Therefore, you must create a DB2 instance with a new name if an existing DB2 instance name is the same as the production DB2 instance name. To access the clone database from a different instance name, you must change the default tablespace container's path name by executing following command on the database server:

   ```
   mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n
   ```

   Where the following variables are defined as:

   – `DBDir` identifies the name assigned to the directory or device on which the database is created.

   – `OldInstanceName` identifies the name assigned to the DB2 instance to which the production database belongs.

   – `NewInstanceName` identifies the name assigned to the DB2 instance to which the clone database belongs.

   For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

   ```
   mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
   ```

2. Change the database name and tablespace containers header information using the db2inidb command as specified in section 1.4. To do this, you will need to create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to that shown in Example 4-10.

Example 4-10 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is *mydb* and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to *mydbcl*. It has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is *db2inst1* and clone database instance name is *db2instc*.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

   ```
   db2start
   ```

   If the production database was offline while the Snapshot copies were created, the clone database becomes available after executing db2start.

   The production database was offline during the Snapshot creation process. Therefore, you can connect to the clone database and start using it.

4. Catalog the source database.

   If the cloned database is accessed from same DB2 instance as the production database on the production database server, then on execution the db2relocatedb command uncatalogs the source database. Therefore, you need to catalog the source database by executing the following command on the database server:

   ```
   db2 "catalog database [DatabaseName] as [DatabaseAlias] on
   [FileSystem]"
   ```

   Where the following variables are defined as:

   – `DatabaseName` identifies the name assigned to the database that is being cataloged.

   – `DatabaseAlias` identifies the alias name assigned to the database that is being cataloged.

   – `FileSystem` specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named mydb that resides on file system named /mnt/dbdata, you would execute the following command on the database server:

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

4.4.7 Catalog the source database

If the cloned database is accessed from same DB2 instance as the production database on the production database server, then, on execution, the **`db2relocatedb`** command uncatalogs the source database. Therefore, you must catalog the source database by executing the following command on the database server:

```
db2 "catalog database [DatabaseName] as [DatabaseAlias] on [FileSystem]"
```

Where the following variables are defined as:

- `DatabaseName` identifies the name assigned to the database that is being cataloged.
- `DatabaseAlias` identifies the alias name assigned to the database that is being cataloged.
- `FileSystem` specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named `mydb` that resides on file system named `/mnt/dbdata`, execute the following command on the database server:

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

4.4.8 Verify the cloned database

After performing the previous steps, you must check the entire database for architectural correctness. You can do so by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

Where `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to test the database named *mydbcl*, execute the following command on the database server:

```
db2dart mydbcl /db
```

The `db2dart` utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the <$HOME>/sqllib/db2dump/DART0000/ directory. Read the summary at the end of the report to see if there are any errors.

If the source and the clone database need to be accessed from the same database server, then you must complete the additional steps described in Appendix A, “Configuring UNIX to access cloned and source databases in an NAS environment” on page 99.

4.5 Clone an online database to a remote storage system

In this section, you learn how to create a clone database on a remote storage system when the source database is online. The step-by-step cloning process is described in the following subsections.

4.5.1 Configure and initialize SnapMirror

The steps necessary to configure and initialize a SnapMirror relationship are described in 4.4.1, “Configure SnapMirror” on page 43 and 4.4.2, “Initialize SnapMirror” on page 45 respectively. After completing these steps, you should have working SnapMirror relationships for the volumes that are used for the database to be cloned.

4.5.2 Bring the source database into a consistent state (suspend writes)

The source database is online. Therefore, to prevent partial page writes while database cloning is in progress, you must suspend the write operations to the database temporarily by executing the following command on the database server:

```
db2 set write suspend for database
```

The **`set write suspend for database`** command causes the DB2 database manager to suspend all write operations to tablespace containers and log files that are associated with the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The cloning process is completed very quickly, so the database does not need to stay in write suspend mode for more than a few seconds.

4.5.3 Create Snapshot copies of the FlexVol volumes

Create a Snapshot copy of each FlexVol volume that is used for the source database by executing the following command on the source storage system:

```
snap create [VolName] [SnapName]
```

Where the following variables are defined as:

- `VolName` identifies the name assigned to the FlexClone volume that is to be created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_cl_snap01` for a FlexVol volume named `dbdata`, you would execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.1
```

It is recommended that you develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purposes.

Important: The Snapshot copies are created on the SnapMirror source storage system.

4.5.4 Resume normal database operations (resume writes)

After the Snapshot copies are created, you must resume write operations to the database by executing the following command on the database server:

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

4.5.5 Update the SnapMirror destination

Update the SnapMirror destination volumes manually by executing the following command on the destination storage system:

```
snapmirror < -S [SourceStorageSystem]:[VolumeName]> update
[DestinationStorageSystem]:[VolumeName]
```

Where the following variables are defined as:

- `SourceStorageSystem` identifies the name assigned to the SnapMirror source storage system.
- `VolumeName` identifies the name assigned to the volume that is being used as the SnapMirror destination.
- `DestinationStorageSystem` identifies the name assigned to the SnapMirror destination storage system.

For example, to update SnapMirror for a volume named `dbdata`, execute the following command on the SnapMirror destination storage system named *dststore*:

```
snapmirror -S srcstore:dbdata update dststore:dbdata
```

Update the SnapMirror relationship for each volume that is used for the database.

When the update command is executed for an asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including Snapshot copies.

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes available on the destination volume immediately; therefore, manual update is not required.

Important: Snapshot copies are also created automatically for SnapMirror update purposes. It is recommended not to use these automatically created Snapshot copies to create a clone volume.

4.5.6 Create clone volumes using Snapshot copies

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the source storage system and replicated to the destination. A clone volume can be created from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined as:

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is the source for the clone volume.

- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone volume named `dbdata_cl` from the Snapshot copy named `dbdata_cl_snp.1` of the volume named `dbdata`, execute the following command on the destination storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.1
```

Important: The Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship will fail.

4.5.7 Create NFS export entries for the cloned volumes

To mount a clone volume to the database server, you must create an export entry for it in the /etc/exports file that resides on the NetApp FAS or IBM N Series storage system. The export entry can be created by executing the following command on the IBM N Series storage system:

```
exportfs -p rw=[HostName],root=[HostName] [PathName]
```

Where the following variables are defined as:

- `HostName` identifies the name assigned to the database server.
- `PathName` identifies the name assigned to the flexible volume.

For example, to create an export entry for a clone volume named `dbdata_cl` and allow root access from the database server named `hostdst` for it, execute the following command on the storage system:

```
exportfs -p rw=hostdst,root=hostdst /vol/dbdata_cl
```

Repeat this step to create an export entry for each clone volume that is used for the clone database.

4.5.8 Mount the cloned volumes

To access the clone database, you must mount the clone volumes to a database server. First, you create a mount point for each clone volume and append a mount entry to the /etc/fstab file. The mount entry should specify the mount options, and it should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

Where the following variables are defined as:

- `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- `FlexVolName` identifies the name assigned to the clone volume.
- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named *dbdata_cl* that resides on a storage system named *srcstore*, append the following entry to the /etc/fstab file on the database sever:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

Where `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has a mount entry specified in the `/etc/fstab file`, execute the following command on the second database server named hostdst:

```
mount /mnt/dbdata_cl
```

The database servers we used had a Linux operating system.

To operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where the following variables are defined as:

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the files system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, execute the following command on the database server named `hostdst`:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

4.5.9 Configure the cloned database

The clone volumes created in the 4.5.6, “Create clone volumes using Snapshot copies” on page 58 and mount in the 4.5.8, “Mount the cloned volumes” on page 59 of this section are used as the storage containers for the cloned database in this section. You can skip the part (A) and (B) of this step if following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is same as the production or source database instance name.
- The mount points that are used to mount the clone volumes have the same name as the mount points that are used to mount volumes of the production database.

1. By default when database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   ```
   [DBDir]/[InstanceName]/NODE000n
   ```

   Where the following variables are defined as:

   – `DBDir` identifies the name assigned to the directory or device on which the database is created.

   – `InstanceName` identifies the name assigned to the DB2 instance to which the database belongs.

   For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace containers will reside in the following directory.

   ```
   /mnt/dbdata/db2inst1/NODE0000
   ```

   For a database server the DB2 instance name has to be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is same as the production DB2 instance name. In order to access the clone database from a different instance name you need to change the default tablespace container's path name by executing following command on the database server:

   ```
   mv [DBDir]/[OldInstanceName]/NODE000n
   [DBDir]/[NewInstanceName]/NODE000n
   ```

Where the following variables are defined as:

– `DBDir` identifies the name assigned to the directory or device on which the database is created.

– `OldInstanceName` identifies the name assigned to the DB2 instance to which the production database belongs.

– `NewInstanceName` identifies the name assigned to the DB2 instance to which the clone database belongs.

For example, to access the clone database from the instance named `db2instc`, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

2. Change the database name and tablespace containers header information using the **db2inidb** command as specified in section 1.4. To do this, you must create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to that shown in Example 4-11.

Example 4-11 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is `mydb` and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to *mydbcl*; it has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is db2inst1 and clone database instance name is db2instc.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing **db2start**.

The production database was online when the Snapshot copies were created. Therefore, you need to use the **db2inidb** utility to initialize the clone database as a snapshot. You can do this by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

Where `DatabaseName` identifies the name of the clone database that is going to be used for the test environment.

For example, to initialize a clone database named *mydbcl*, execute the following commands on the database server:

```
db2inidb mydbcl as snapshot
```

4.5.10 Verify the cloned database

After performing the previous steps, you must check the entire database for architectural correctness. You can do so by executing the following command on the database server:

```
db2dart [DatabaseName] /db
```

Where `DatabaseName` identifies the name of the clone database used for the test environment.

For example, to test the database named *mydbcl*, execute the following command on the database server:

```
db2dart mydbcl /db
```

The **db2dart** utility inspects the entire database for architectural correctness and generates a detailed report. The report is generated in the <$HOME>/sqllib/db2dump/DART0000/ directory. Read the summary at the end of the report and check to see if there are any errors.

If the source and the clone database must be accessed from the same database server, then you must complete the additional steps described in Appendix A, “Configuring UNIX to access cloned and source databases in an NAS environment” on page 99.

5

Cloning a DB2 UDB database in the SAN environment

This chapter discusses cloning DB2 databases in SAN environments.

5.1 Clone an offline database on the same storage system

To clone a DB2 database that is offline, complete the steps in this section.

5.1.1 Bring the source database offline

Bring the database offline by terminating all the application connections to the database that is to be cloned by executing the following command on the database server:

```
db2 force applications all
```

For example, to terminate all application connections, you would execute the following command on the database server:

```
db2 force applications all
```

5.1.2 Create Snapshot copies of the FlexVol volumes

Create a Snapshot copy of each volume that is used for the database by executing the following command on the source storage system:

```
snap create [VolName] [SnapName]
```

Where the following variables are defined as:

- `VolName` identifies the name assigned to the FlexClone volume that is to be created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named dbdata_snap.01for a FlexVol volume named dbdata, you would execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.01
```

It is recommended that you develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purposes.

5.1.3 Start the source database

After the Snapshot copies of the FlexVol volumes of the source database are created, you can connect to the source database and start using it.

5.1.4 Create clone volumes using Snapshot copies

After the Snapshot copies are created, you must create a clone of each FlexVol volume that contains the database's data or transaction log files by executing the following command on the storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined as:

- `CloneVol` identifies the name assigned to the new clone volume.
- `ParentVol` identifies the name assigned to the FlexVol volume that is to be cloned.
- `ParentSnap` identifies the name assigned to the snapshot copy that is used as the base for the clone volume.

  For example, to create a clone volume of a FlexVol volume named *dblogs* using the Snapshot copy named `dbdata_cl_snp.01`, execute the following command on the storage system:

  ```
  vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.01
  ```

  Note: A Snapshot copy is not required to create a clone of a FlexVol volume. If you do not explicitly create a Snapshot copy and specify it when executing the vol clone command, a Snapshot copy will be implicitly created and used for the clone volume. A Snapshot copy created implicitly will have a system assigned name. We recommend explicitly creating a Snapshot and assigning it a meaningful name before creating a clone FlexVol volume.

  For example, to create a clone volume of a FlexVol volume named dbdata, you would execute the following command on the storage system:

  ```
  vol clone create dbdata_cl -s volume -b dbdata
  ```

  A Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone exists.

5.1.5 Create new mapping for the LUNs.

When a clone volume is created, by default, LUNs residing on it have the same mapping as the source FlexVol volume LUNs, and they are placed offline. You can see the LUN status by executing the following command on the storage system:

```
lun show
```

The output from the **lun show** command should look similar to that shown in Example 5-1.

Example 5-1 Output from the lun show command

```
lun show
/vol/dbdata/data    15g (16106127360)  (r/o, online, mapped)
/vol/dbdata_cl/data 15g (16106127360)  (r/w, offline, mapped)
```

The LUN mappings can be listed by executing the following command on the storage system:

```
lun show -m
```

The output from the **lun show -m** command should look similar to that shown in Example 5-2.

Example 5-2 Output from the lun show -m command

```
lun show -m
LUN path                 Mapped to         LUN ID  Protocol
-------------------------------------------------------------
/vol/dbdata/data           host_src_fcp_igp      0     FCP
/vol/dbdata_cl/data        host_src_fcp_igp      0     FCP
```

You can see from the output that the LUNs on the clone volume have the same mapping as the parent.

The FlexClone volume LUNs can be accessed from the same database server used to access the source database, or from a completely different server. The scenarios described in this paper were produced using a second database sever to access the clone database.

To access the clone database from a second database server, you must complete the following steps:

1. Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

   ```
   lun unmap [LunPath]  [iGroupName]
   ```

   Where the following variables are defined as:

   – `LunPath` identifies the name assigned to the new clone volume.

   – `GroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

For example, to remove old mapping for a LUN named `/vol/dbdata_cl/data` from an igroup named `host_src_fcp_igp`, execute the following command on the storage system:

```
lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
```

You should see a message similar to that shown in Example 5-3.

Example 5-3 Messages from lun unmap

```
Thu Aug  3 09:37:10 MST [lun.map.unmap:info]: LUN
/vol/voldbdata_cl/data unmapped from
initiator group host_src_fcp_igp
```

2. Create a new mapping for the LUNs on the clone volume by using a new ID and, mapping them to a new igroup, or both. A LUN mapping can be created by executing the following command on the storage system:

   ```
   lun map [LunPath]  [iGroupName] [LunId]
   ```

   Where the following variables are defined as:

   – `LunPath` identifies the name assigned to the new clone volume.

   – `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

   – `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

   For example, to map a LUN named /vol/dbdata_cl/data to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

   ```
   lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
   ```

   You will see messages similar to that shown in Example 5-4.

Example 5-4 Message from lun map

```
lun map: auto-assigned host_dst_fcp_igp=4
```

3. After remapping, each LUN that is on the clone volume and that is used for the clone database needs to be brought online by executing the following command on the storage system:

   ```
   lun online [LunPath]
   ```

   Where `LunPath` identifies the name assigned to the new clone volume.

   For example, to bring a LUN named `/vol/dbdata_cl/data` online, execute the following command on the storage system:

   ```
   lun online /vol/dbdata_cl/data
   ```

5.1.6 Mount the FlexClone volume LUNs

To mount the FlexClone volume LUNs, follow these steps:

1. Refresh the HBA driver on the database server. For example, to refresh a Qlogic FC HBA on a Linux database host, you would execute the following commands on the database server:

   ```
   modprobe -r qla2300
   modprobe -v qla2300
   ```

   You should see messages similar to that shown in Example 5-5.

Example 5-5 Messages from modprobe

```
insmod /lib/modules/2.6.9-34.ELsmp/kernel/drivers/scsi/scsi_transport_fc.ko
insmod /lib/modules/2.6.9-34.ELsmp/kernel/drivers/scsi/qla2xxx/qla2xxx.ko
insmod /lib/modules/2.6.9-34.ELsmp/kernel/drivers/scsi/qla2xxx/qla2300.ko
```

   For any other operating system (OS) and HBA, refer to the OS reference manual and the HBA installation guide.

2. Obtain the device names for the LUNs by executing the following command on the database server that is to be used to access the clone database:

   ```
   sanlun lun show
   ```

   You should receive a output similar to Example 5-6.

Example 5-6 Output sanlun lun show command

```
sanlun lun show
filer:    lun-pathname        device   filename adapter Protocol lun size      lun state
itsotuc2: /vol/db2data/lun    hdisk14 fcs1     FCP     39.9g  (42858446848)  GOOD
itsotuc2: /vol/db2datanfs/lun hdisk16 fcs1     FCP     8.0g   (8571060224)   GOOD
itsotuc2: /vol/db2logs/lun    hdisk15 fcs1     FCP     23.9g  (25715277824)  GOOD
itsotuc2: /vol/db2datacl/lun  hdisk18 fcs1     FCP     39.9g  (42858446848)  GOOD
itsotuc2: /vol/db2logsnfs/lun hdisk17 fcs1     FCP     8.0g   (8571060224)   GOOD
itsotuc2: /vol/db2logscl/lun  hdisk19 fcs1     FCP     23.9g  (25715277824)  GOOD
```

3. Mount LUN devices by executing the following command on the database server:

   ```
   mount [DeviceName] [MountPoint]
   ```

   Where the following variables are defined as:

   – `DeviceName` identifies the name assigned to the new clone volume.

   – `MountPoint` identifies the name assigned to the mount point that is used to mount the LUN device.

For example, to mount a LUN device that is identified by the name /dev/sdb by the database server, execute the following command on the database server:

```
mount /dev/sdb /mnt/dbdata
```

4. To operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. Ownership can be changed by executing the following command on the database server:

   ```
   chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
   ```

   Where the following variables are defined as:

   – `InstanceOwner` identifies the name assigned to the user who owns the database instance.

   – `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.

   – `FileSystem` identifies the name of the file system whose ownership is changed.

   For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, execute the following command on the database server named *hostdst*:

   ```
   chown -R db2inst1:db2adm /mnt/dbdata_cl
   ```

5.1.7 Configure the cloned database

You use the LUNs that resides on the FlexClone volume that you created in 5.1.4, “Create clone volumes using Snapshot copies” on page 67 and mounted in 5.1.6, “Mount the FlexClone volume LUNs” on page 70 as the storage containers for the cloned database. You can skip steps 1 and 2 if following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is same as the production or source database instance name.
- The mount points that are used to mount the LUNs on the FlexClone volumes have the same name as the mount points that are used to mount the LUNs on the FlexVol volumes used for the production database.

To configure the cloned database, follow these steps:

1. By default when database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   ```
   [DBDir]/[InstanceName]/NODE000n
   ```

Where the following variables are defined as:

- `DBDir` identifies the name assigned to the directory or device on which the database is created.
- `InstanceName` identifies the name assigned to the DB2 instance to which the database belongs.

  For example, if the DB2 instance name is *db2inst1* and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory.

  ```
  /mnt/dbdata/db2inst1/NODE0000
  ```

For a database server, the DB2 instance name must be unique. Therefore, you must create a DB2 instance with a new name if an existing DB2 instance name is same as the production DB2 instance name. In order to access the clone database from a different instance name, you must change the default tablespace container's path name by executing following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n
```

Where the following variables are defined as:

- `DBDir` identifies the name assigned to the directory or device on which the database is created.
- `OldInstanceName` identifies the name assigned to the DB2 instance to which the production database belongs.
- `NewInstanceName` identifies the name assigned to the DB2 instance to which the clone database belongs.

For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

2. Change the database name and tablespace containers header information using the **`db2inidb`** command as specified in 1.5, “The db2inidb command” on page 6. To do this, you must create a configuration file that identifies the source database information and specifies the new clone database information.

A configuration file for this scenario should look similar to that shown in Example 5-7.

Example 5-7 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is `mydb` and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to *mydbcl*. It has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is *db2inst1* and clone database instance name is *db2instc*.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

   ```
   db2start
   ```

   If the production database was offline while the Snapshot copies were created, the clone database becomes available after executing **db2start**.

   The production database was offline during the Snapshot creation process. Therefore, you can connect to the clone database and start using it.

After these steps, your database is ready to use and you can connect and verify the database.

If the source and the clone database need to be accessed from the same database server, then you must complete the additional steps described in the Appendix B, “Configuring UNIX to access cloned and source databases in an SAN environment” on page 105.

In this scenario, the LUNs on the clone volume are accessed using FCP, but they can be accessed using the iSCSI access protocol, as well.

5.2 Clone an online database on the same storage system

To create a clone on an online DB2 database on the same storage system in a SAN environment, complete the steps in this section.

5.2.1 Bring the source database into a consistent state (suspend writes)

In this scenario, the source database is online. Therefore, to prevent partial page writes while database cloning is in progress, the write operations to the database need to be temporarily suspended by executing the following command on the database server:

```
db2 set write suspend for database
```

The **`set write suspend for database`** command causes the DB2 database manager to suspend all write operations to the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The FlexVol volume clone process is completed very quickly, so the database doesn't need to stay in write suspend mode for more than a few seconds.

5.2.2 Create Snapshot copies of the FlexVol volumes

The LUNs reside within a FlexVol volume. Therefore, you need to create a Snapshot copy for each FlexVol volume that consists of LUNs that are used for the production database. A Snapshot copy of a FlexVol volume can be created by executing the following command on the storage system

```
snap create [VolName] [SnapName]
```

Where the following variables are defined as:

- `VolName` identifies the name assigned to the FlexClone volume that is to be created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_snap01` for a FlexVol volume named *dbdata*, execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.1
```

It is recommended that you develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purposes.

5.2.3 Resume normal database operation (resume writes)

After you create Snapshot copies, you must resume write operations to the database by executing the following command on the database server:

```
set write resume for database
```

5.2.4 Clone the FlexVol volumes using Snapshot copies

Next, create a clone of each FlexVol volume by using the Snapshot copies created in step 5.2.2, "Create Snapshot copies of the FlexVol volumes" on page 74. A clone volume can be created by executing the following command on the storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined as:

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is the source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone of a FlexVol volume named *dblogs* by using its existing Snapshot copy named `dbdata_cl_snp.1`, execute the following command on the storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.1
```

The Snapshot copy that is used as the base for the clone volume cannot be deleted as long as the clone volume exists.

5.2.5 Create new mapping for the LUNs

As noted earlier, a clone volume is a writable Snapshot copy, and at the time of clone creation, it has data that is exactly similar to that of the source volume. The LUNs on the clone volume have the same mapping as the LUNs on the source volume, and they are set offline. To create new mapping for LUNs on the FlexClone volumes, complete following steps:

1. Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

   ```
   lun unmap [LunPath]  [iGroupName]
   ```

Where the following variables are defined as:

– `LunPath` identifies the name assigned to the new clone volume.

– `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

For example, to remove old mapping for a LUN named `/vol/dbdata_cl/data` from an igroup named `host_src_fcp_igp`, execute the following command on the storage system:

```
lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
```

2. Create a new mapping for the LUNs on the clone volume by using a new ID, r by mapping them to a new igroup, or both. Create a LUN mapping by executing the following command on the storage system:

   ```
   lun unmap [LunPath]  [iGroupName] [LunId]
   ```

   Where the following variables are defined as:

   – `LunPath` identifies the name assigned to the new clone volume.

   – `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

   – `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

   For example, to map a LUN named `/vol/dbdata_cl/data` to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

   ```
   lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
   ```

3. After remapping, each LUN that is on the clone volume and that is used for the clone, the database needs to be brought online by executing the following command on the storage system:

   ```
   lun online [LunPath]
   ```

   Where `LunPath` identifies the name assigned to the new clone volume.

   For example, to bring a LUN named `/vol/dbdata_cl/data` online, execute the following command on the storage system:

   ```
   lun online /vol/dbdata_cl/data
   ```

5.2.6 Mount the LUNs that reside on the FlexClone volumes

To mount the LUNs that reside on the FlexClone volumes, complete following steps:

1. Refresh the HBA driver on the database server. For example, to refresh a Qlogic FC HBA on a Linux database host, execute the following commands on the database server:

   ```
   modprobe -r qla2300
   modprobe -v qla2300
   ```

   For any other operating system (OS) and HBA, refer to the OS reference manual and the HBA installation guide.

2. Obtain the LUN device names by executing the following command on the database server that is to be used to access the clone database:

   ```
   sanlun lun show
   ```

3. Mount the LUN devices by executing the following command on the database server:

   ```
   mount [DeviceName] [MountPoint]
   ```

   Where the following variables are defined as:

   – `DeviceName` identifies the name assigned to the new clone volume.

   – `MountPoint` identifies the name assigned to the mount point that is used to mount the LUN device.

   For example, to mount a LUN device that is identified by the name /dev/sdb by the database server, you would execute the following command on the database server:

   ```
   mount /dev/sdb /mnt/dbdata
   ```

4. To operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. Ownership can be changed by executing the following command on the database server:

   ```
   chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
   ```

   Where the following variables are defined as:

   – `InstanceOwner` identifies the name assigned to the user who owns the database instance.

   – `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.

   – `FileSystem` identifies the name of the files system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, execute the following command on the database server named *hostdst*:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

5.2.7 Configure the cloned database

You use the clone volumes that you created in 5.2.4, "Clone the FlexVol volumes using Snapshot copies" on page 75 and mounted in 5.2.6, "Mount the LUNs that reside on the FlexClone volumes" on page 77 as the storage containers for the cloned database. You can skip steps 1 and 2 if following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is same as the production or source database instance name.
- The mount points that are used to mount the clone volumes have same name as the mount points that are used to mount volumes of the production database.

To configure the cloned database, follow these steps:

1. By default when database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   ```
   [DBDir]/[InstanceName]/NODE000n
   ```

   Where the following variables are defined as:

   – `DBDir` identifies the name assigned to the directory/device the database is created on.

   – `InstanceName` identifies the name assigned to the DB2 instance to which the database belongs.

   For example, if the DB2 instance name is db2inst1 and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory.

   ```
   /mnt/dbdata/db2inst1/NODE0000
   ```

   For a database server the DB2 instance name has to be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is same as the production DB2 instance name. In order to access the clone database from a different instance name you need to change the default tablespace container's path name by executing following command on the database server:

   ```
   mv [DBDir]/[OldInstanceName]/NODE000n [DBDir]/[NewInstanceName]/NODE000n
   ```

Where:

- `DBDir` identifies the name assigned to the directory or device to which the database is created.
- `OldInstanceName` identifies the name assigned to the DB2 instance to which the production database belongs.
- `NewInstanceName` identifies the name assigned to the DB2 instance to which the clone database belongs.

For example, to access the clone database from the instance named `db2instc`, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

2. Change the database name and tablespace containers header information using the **`db2inidb`** command as specified in 1.5, “The db2inidb command” on page 6. To do this, create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to that shown in Example 5-8.

Example 5-8 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is *mydb* and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to *mydbcl*. It has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is *db2inst1* and clone database instance name is *db2instc*.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

```
db2start
```

If the production database was offline during the Snapshot copy process, the clone database becomes available after executing **`db2start`**.

The production database was online when the Snapshot copies were created. Therefore, you need to use the **db2inidb** utility to initialize the clone database as a snapshot. You can do this by executing the following command on the database server:

```
db2inidb database [DatabaseName] as snapshot
```

Where `DatabaseName` identifies the name of the clone database that is going to be used for the test environment.

For example, to initialize a clone database named *mydbcl*, execute the following commands on the database server:

```
db2inidb mydbcl as snapshot
```

After you perform these steps, the clone database is ready for use. You can connect to the database and perform data verification.

If the source and the clone database need to be accessed from the same database host, then complete the additional steps described in Appendix B, “Configuring UNIX to access cloned and source databases in an SAN environment” on page 105.

In this scenario, the LUNs on the clone volume are accessed using FCP, but they can be accessed using the iSCSI access protocol as well.

5.3 Clone an offline database to a remote storage system

To create a clone on the second storage system at a remote location, complete the steps in this section.

5.3.1 Configure and initialize SnapMirror

The steps necessary to set up and initialize a SnapMirror relationship are described in 4.4.1, “Configure SnapMirror” on page 43 and 4.4.2, “Initialize SnapMirror” on page 45 respectively. After completing these steps, you should have working SnapMirror relationships for the volumes that are used for the database to be cloned.

In the SnapMirror relationship, the destination volumes are read-only. Therefore, to create a clone of the destination volume, you need a Snapshot copy that is created on the SnapMirror source and is replicated to the destination volume. Such a Snapshot copy can be obtained by completing the steps in this section.

5.3.2 Bring the database offline

Terminate all the application connections to the database by executing the following command on the database server:

```
db2 force applications all
```

5.3.3 Create Snapshot copies of the FlexVol volumes

The LUNs reside within a FlexVol volume. Therefore, create a Snapshot copy for each FlexVol volume that consists of LUNs that are used for the production database. A Snapshot copy of a FlexVol volume can be created by executing the following command on the storage system:

```
snap create [VolName] [SnapName]
```

Where the following variables are defined as:

- *VolName* identifies the name assigned to the FlexClone volume that is to be created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named *dbdata_snap01* for a FlexVol volume named *dbdata*, execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.1
```

It is recommended that you develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purpose.

5.3.4 Update the SnapMirror destination volumes

Update the SnapMirror destination volumes manually by executing the following command on the destination storage system:

```
snapmirror < -S [SourceStorageSystem]:[VolumeName]> update
[DestinationStorageSystem]:[VolumeName]
```

Where the following variables are defined as:

- `SourceStorageSystem` identifies the name assigned to the SnapMirror source storage system.
- `VolumeName` identifies the name assigned to the SnapMirror source or destination volume.
- `DestinationStorageSystem` identifies the name assigned to the SnapMirror destination storage system.

For example, to update SnapMirror for a volume named *dbdata*, execute the following command on the SnapMirror destination storage system named *dststore*:

```
snapmirror -S srcstore:dbdata update dststore:dbdata
```

Update the SnapMirror relationship for each volume that is used for the database.

For asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including Snapshot copies.

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes visible on the destination volume immediately; therefore, manual update is not required.

Important: The Snapshot copies are also created automatically for SnapMirror update purposes. It is recommended not to use these automatically created Snapshot copies to create a clone volume.

5.3.5 Create FlexClone volumes using Snapshot copies

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the SnapMirror source storage system and replicated to the destination. A clone volume can be created from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined as:

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as the source for the clone volume.

For example, to create a clone volume named *dbdata_cl* from the Snapshot copy named `dbdata_cl_snp.1` of the volume named *dbdata*, execute the following command on the destination storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.1
```

Important: The Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship will fail.

5.3.6 Create new mapping for LUNs that reside on the clone volumes

When a clone volume is created, by default, LUNs residing on it have the same mapping as the source FlexVol volume LUNs, and they are placed offline. You can see the LUN status by executing the following command on the storage system:

```
lun show
```

The output from the **`lun show`** command should look similar to that shown in Example 5-9.

Example 5-9 Output from the lun show command

```
lun show
/vol/dbdata/data    15g (16106127360)  (r/o, online, mapped)
/vol/dbdata_cl/data 15g (16106127360)  (r/w, offline, mapped)
```

The LUN mappings can be listed by executing the following command on the storage system:

```
lun show -m
```

The output from the **`lun show -m`** command should look similar to that shown in Example 5-10.

Example 5-10 Output from the lun show -m command

```
lun show -m
LUN path                 Mapped to          LUN ID  Protocol
--------------------------------------------------------------
/vol/dbdata/data           host_src_fcp_igp      0     FCP
/vol/dbdata_cl/data        host_src_fcp_igp      0     FCP
```

You can see from the output that the LUNs on the clone volume have the same mapping as the parent.

The FlexClone volume LUNs can be accessed from the same database server that is used to access the source database, or from a completely different server. The scenarios described in this paper were produced using a second database sever to access the clone database.

To access the clone database from a second database server, complete the following steps:

1. Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

   ```
   lun unmap [LunPath]  [iGroupName]
   ```

   Where the following variables are defined as:

   – `LunPath` identifies the name assigned to the new clone volume.

   – `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

   For example, to remove the old mapping for a LUN named `/vol/dbdata_cl/data` from an igroup named `host_src_fcp_igp`, execute the following command on the storage system:

   ```
   lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
   ```

2. Create a new mapping for the LUNs on the clone volume by using a new ID and by mapping them to a new igroup. A LUN mapping can be created by executing the following command on the storage system:

   ```
   lun unmap [LunPath]  [iGroupName] [LunId]
   ```

   Where the following variables are defined as:

   – `LunPath` identifies the name assigned to the new clone volume.

   – `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

   – `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

   For example, to map a LUN named `/vol/dbdata_cl/data` to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

   ```
   lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
   ```

3. After remapping, each LUN that is on the clone volume and that is used for the clone database must be brought online by executing the following command on the storage system:

   ```
   lun online [LunPath
   ```

   Where `LunPath` identifies the name assigned to the new clone volume.

   For example, to bring a LUN named /vol/dbdata_cl/data online, you would execute the following command on the storage system:

   ```
   lun online /vol/dbdata_cl/data
   ```

5.3.7 Mount the FlexClone volume LUNs

To mount the LUNs that reside on the FlexClone volumes, complete following steps:

1. Refresh the HBA driver on the database server. For example, to refresh a Qlogic FC HBA on a Linux database host, execute the following commands on the database server:

   ```
   modprobe -r qla2300
   modprobe -v qla2300
   ```

   For any other operating system (OS) and HBA, refer to the OS reference manual and the HBA installation guide.

2. Obtain the device names for the LUNs by executing the following command on the database server that is to be used to access the clone database:

   ```
   sanlun lun show
   ```

3. Mount LUN devices by executing the following command on the database server:

   ```
   mount [DeviceName] [MountPoint]
   ```

   Where the following variables are defined as:

   – `DeviceName` identifies the name assigned to the new clone volume.

   – `MountPoint` identifies the name assigned to the mount location that is used to mount the LUN device.

   For example, to mount a LUN device named /dev/sdb to a mount location named /mnt/dbdata_cl, execute the following command on the database server:

   ```
   mount /dev/sdb /mnt/dbdata_cl
   ```

4. In order to operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. Change the ownership by executing the following command on the database server:

   ```
   chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
   ```

   Where the following variables are defined as:

   – `InstanceOwner` identifies the name assigned to the user who owns the database instance.

   – `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.

   – `FileSystem` identifies the name of the files system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named /mnt/dbdata_cl, execute the following command on the database server named *hostdst*:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

5.3.8 Renaming the cloned database

Rename the clone database and update the tablespace containers header information. You are going to use the LUNs that reside on the FlexClone volume that you created and mounted as the storage containers for the cloned database. You can skip steps 1 and 2 if the following two conditions are true for your environment:

- The name of the DB2 instance that you used for the clone database is same as the production or source database instance name.
- The mount points that are used to mount the LUNs on the FlexClone volumes have the same name as the mount points that are used to mount the LUNs on the FlexVol volumes used for the production database.

To rename the cloned database, follow these steps:

1. By default when database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   [DBDir]/[InstanceName]/NODE000n

   where:

   – DBDir identifies the name assigned to the directory/device the database is created on.

   – InstanceName identifies the name assigned to the DB2 instance the database belongs to.

   For example, if the DB2 instance name is db2inst1 and the database was created on the directory named /mnt/dbdata, the default tablespace containers will reside in the following directory.

   /mnt/dbdata/db2inst1/NODE0000

   For a database server the DB2 instance name has to be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is same as the production DB2 instance name. In order to access the clone database from a different instance name you need to change the default tablespace container's path name by executing following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n
[DBDir]/[NewInstanceName]/NODE000n
```

where:

- DBDir identifies the name assigned to the directory/device the database is created on.
- OldInstanceName identifies the name assigned to the DB2 instance the production database belongs to.
- NewInstanceName identifies the name assigned to the DB2 instance the clone database belongs to.

For example, to access the clone database from the instance named db2instc, you would execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

2. Now you need to change the database name and tablespace containers header information using the **db2inidb** command as specified in 1.5, "The db2inidb command" on page 6. To do this, you need to create a configuration file that identifies the source database information and specifies the new clone database information. A sample configuration file for this scenario should look similar to that shown in Example 5-11.

Example 5-11 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is *mydb* and it has its logs on /mnt/dblogs and data on /mnt/dbdata . The clone database is to be renamed to *mydbcl*. It has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is *db2inst1* and clone database instance name is *db2instc*.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

   ```
   db2start
   ```

If the production database was offline while the Snapshot copies were created, the clone database becomes available after executing db2start.

The production database was offline during the Snapshot creation process. Therefore, you can connect to the clone database and start using it.

If the source and the clone database need to be accessed from the same database host, then you need to complete the additional steps described in the Appendix B, "Configuring UNIX to access cloned and source databases in an SAN environment" on page 105.

In this scenario, the LUNs on the clone volume are accessed using FCP, but they can be accessed using the iSCSI access protocol as well.

5.4 Clone an online database to a remote storage system

In this section, you learn how to create a clone database on the remote storage system when the source database is online. The step-by-step cloning process is described in the following subsections.

5.4.1 Set up and initialize SnapMirror

The steps necessary to set up and initialize a SnapMirror relationship are described in 4.4.1, "Configure SnapMirror" on page 43 and 4.4.2, "Initialize SnapMirror" on page 45 respectively. After completing the steps described in these sections, you should have working SnapMirror relationships for the volumes that are used for the database to be cloned.

In the SnapMirror relationship, the destination volumes are read-only. Therefore, to create a clone of the destination volume, you need a Snapshot copy that is created on the SnapMirror source and that is replicated to the destination volume. Such a Snapshot copy can be obtained by completing the following steps.

5.4.2 Bring the database into a consistent state (suspend writes)

In this scenario, the source database is online. Therefore, to prevent partial page writes while the database Snapshot copy process is in progress, the write operations to the database need to be temporarily suspended by executing the following command on the database server:

```
   db2 set write suspend for database
```

The SET WRITE SUSPEND FOR DATABASE command causes the DB2 database manager to suspend all write operations to the current database. Read-only transactions continue uninterrupted, provided that they do not request a resource that is being held by the suspended I/O process. The FlexVol volume clone process is completed very quickly, so the database doesn't need to stay in write suspend mode for more than a few seconds

5.4.3 Create Snapshot copies

Create Snapshot copies of the volumes that are used for the database. The LUNs reside within a FlexVol volume. Therefore, you must create a Snapshot copy for each FlexVol volume that consists of LUNs that are used for the production database. A Snapshot copy of a FlexVol volume can be created by executing the following command on the storage system:

```
snap create [VolName] [SnapName]
```

Where the following variables are defined as:

- `VolName` identifies the name assigned to the FlexClone volume that is to be created.
- `SnapName` identifies the name that is assigned to the Snapshot copy.

For example, to create a Snapshot copy named `dbdata_snap01` for a FlexVol volume named *dbdata*, execute the following command on the storage system:

```
snap create dbdata dbdata_cl_snp.1
```

It is recommended that you develop a naming convention and assign a meaningful name to the Snapshot copies that are created for cloning purpose.

5.4.4 Resume normal operations for the database (resume writes)

After you create Snapshot copies, you must resume write operations to the database by executing the following command on the database server:

```
set write resume for database
```

5.4.5 Update the SnapMirror destination volumes

Update the SnapMirror destination volumes manually by executing the following command on the destination storage system:

```
snapmirror < -S [SourceStorageSystem]:[VolumeName]> update
[DestinationStorageSystem]:[VolumeName]
```

Where the following variables are defined as:

- `SourceStorageSystem` identifies the name assigned to the SnapMirror source storage system.
- `VolumeName` identifies the name assigned to the volume that is being used as the SnapMirror destination.
- `DestinationStorageSystem` identifies the name assigned to the SnapMirror destination storage system.

For example, to update SnapMirror for a volume named *dbdata*, execute the following command on the SnapMirror destination storage system named *dststore*:

```
snapmirror -S srcstore:dbdata update dststore:dbdata
```

Update the SnapMirror relationship for each volume that is used for the database.

For asynchronous SnapMirror, an update is immediately started from the source to the destination to update the destination volume with the contents of the source volume, including Snapshot copies.

For synchronous SnapMirror, the Snapshot copy created on the source volume becomes visible on the destination volume immediately; therefore, manual update is not required.

Important: The Snapshot copies are also created automatically for SnapMirror update purposes. It is recommended that you not use these automatically created Snapshot copies to create a clone volume.

5.4.6 Create clone volumes using Snapshot copies

After the SnapMirror update, each destination volume has a Snapshot copy that was created on the SnapMirror source storage system and replicated to the destination. A clone volume can be created from the Snapshot copy by executing the following command on the destination storage system:

```
vol clone create [CloneVol] -s volume -b [ParentVol] <ParentSnap>
```

Where the following variables are defined as:

- `CloneVol` identifies the name of the FlexClone volume that is being created.
- `ParentVol` identifies the name of the FlexVol volume that is source for the clone volume.
- `ParentSnap` identifies the name of the parent FlexVol volume Snapshot copy that is used as source for the clone volume.

For example, to create a clone volume named *dbdata_cl* from the Snapshot copy named *dbdata_cl_snp.1* of the volume named *dbdata*, execute the following command on the destination storage system:

```
vol clone create dbdata_cl -s volume -b dbdata dbdata_cl_snp.1
```

Important: The Snapshot copy on the SnapMirror source storage system should not be deleted; otherwise, the SnapMirror relationship will fail.

5.4.7 Create new mapping for LUNs that reside on the clone volumes

When a clone volume is created, by default, LUNs residing on it have the same mapping as the source FlexVol volume LUNs, and they are placed offline. You can see the LUN status by executing the following command on the storage system:

```
lun show
```

The output from the **`lun show`** command should look similar to Example 5-12.

Example 5-12 Output from the lun show command

```
lun show
/vol/dbdata/data    15g (16106127360)  (r/o, online, mapped)
/vol/dbdata_cl/data 15g (16106127360)  (r/w, offline, mapped)
```

The LUN mappings can be listed by executing the following command on the storage system:

```
lun show -m
```

The output from the **lun show -m** command should look similar to Example 5-13.

Example 5-13 Output from the lun show -m command

```
lun show -m
LUN path                 Mapped to          LUN ID  Protocol
-------------------------------------------------------------
/vol/dbdata/data           host_src_fcp_igp      0     FCP
/vol/dbdata_cl/data        host_src_fcp_igp      0     FCP
```

You can see from the output that the LUNs on the clone volume have the same mapping as the parent.

The FlexClone volume LUNs can be accessed from the same database server that is used to access the source database, or from a completely different server. The scenarios described in this paper were produced using a second database sever to access the clone database.

In order to access the clone database from a second database server, complete the following steps:

1. Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

   ```
   lun unmap [LunPath]  [iGroupName]
   ```

   Where the following variables are defined as:

   – `LunPath` identifies the name assigned to the new clone volume.

   – `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

   For example, to remove the old mapping for a LUN named /vol/dbdata_cl/data from an igroup named host_src_fcp_igp, you would execute the following command on the storage system:

   ```
   lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
   ```

2. Create a new mapping for the LUNs on the clone volume by using a new ID, mapping them to a new igroup, or both. A LUN mapping can be created by executing the following command on the storage system:

   ```
   lun unmap [LunPath]  [iGroupName] [LunId]
   ```

   Where the following variables are defined as:

   – `LunPath` identifies the name assigned to the new clone volume.

   – `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

   - `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

   For example, to map a LUN named `/vol/dbdata_cl/data` to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

   ```
   lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
   ```

3. After remapping, each LUN that is on the clone volume and that is used for the clone database must be brought online by executing the following command on the storage system:

   ```
   lun online [LunPath]
   ```

   Where `LunPath` identifies the name assigned to the new clone volume.

   For example, to bring a LUN named `/vol/dbdata_cl/data` online, execute the following command on the storage system:

   ```
   lun online /vol/dbdata_cl/data
   ```

5.4.8 Mount the LUN devices

In order to mount the LUNs that reside on the FlexClone volumes, complete following steps:

1. Refresh the HBA driver on the database server. For example, to refresh a Qlogic FC HBA on a Linux database host, execute the following commands on the database server:

   ```
   modprobe -r qla2300
   modprobe -v qla2300
   ```

   For any other operating system (OS) and HBA, refer to the OS reference manual and the HBA installation guide.

2. Obtain the device names for the LUNs by executing the following command on the database server that is to be used to access the clone database:

   ```
   sanlun lun show
   ```

3. Mount LUN devices by executing the following command on the database server:

   ```
   mount [DeviceName] [MountPoint]
   ```

   Where the following variables are defined as:

   - `DeviceName` identifies the name assigned to the new clone volume.
   - `MountPoint` identifies the name assigned to the mount location that is used to mount the LUN device.

For example, to mount a LUN device named /dev/sdb to a mount location named /mnt/dbdata_cl, execute the following command on the database server:

```
mount /dev/sdb /mnt/dbdata_cl
```

4. In order to operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. You can change the ownership by executing the following command on the database server:

   ```
   chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
   ```

   Where the following variables are defined as:

   – `InstanceOwner` identifies the name assigned to the user who owns the database instance.

   – `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.

   – `FileSystem` identifies the name of the files system whose ownership is changed.

   For example, to change ownership of the file system mounted on the mount point named /mnt/dbdata_cl, execute the following command on the database server named *hostdst*:

   ```
   chown -R db2inst1:db2adm /mnt/dbdata_cl
   ```

5.4.9 Configure the cloned database

You use the clone volumes that you created as the storage containers for the cloned database. You can skip steps 1 and 2 if following two conditions are true for your environment:

- The name of the DB2 instance used for the clone database is same as the production or source database instance name.
- The mount points that are used to mount the clone volumes have same name as the mount points that are used to mount volumes of the production database.

To configure the cloned database, follow these steps:

1. By default when database is created, the tablespaces containers for the default tablespaces (SYSCATSPACE, TEMPSPACE1, and USERSPACE1) reside in the following directory:

   ```
   [DBDir]/[InstanceName]/NODE000n
   ```

Where the following variables are defined as:

– `DBDir` identifies the name assigned to the directory or device on which the database is created.

– `InstanceName` identifies the name assigned to the DB2 instance to which the database belongs.

For example, if the DB2 instance name is `db2inst1` and the database was created on the directory named `/mnt/dbdata`, the default tablespace containers will reside in the following directory.

```
/mnt/dbdata/db2inst1/NODE0000
```

For a database server the DB2 instance name must be unique. Therefore, you have to create a DB2 instance with a new name if an existing DB2 instance name is same as the production DB2 instance name.

To access the clone database from a different instance name you must change the default tablespace container's path name by executing the following command on the database server:

```
mv [DBDir]/[OldInstanceName]/NODE000n
[DBDir]/[NewInstanceName]/NODE000n
```

Where the following variables are defined as:

– `DBDir` identifies the name assigned to the directory or device on which the database is created.

– `OldInstanceName` identifies the name assigned to the DB2 instance to which the production database belongs.

– `NewInstanceName` identifies the name assigned to the DB2 instance to which the clone database belongs.

For example, to access the clone database from the instance named *db2instc*, execute the following command on the database server to change the path:

```
mv /mnt/dbdata_cl/db2inst1 /mnt/dbdata_cl/db2instc
```

2. Change the database name and tablespace containers header information using the **`db2inidb`** command as specified in 1.5, “The db2inidb command” on page 6. To do this, you must create a configuration file that identifies the source database information and specifies the new clone database information.

A configuration file for this scenario should look similar to Example 5-14.

Example 5-14 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1,db2instc
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

In this configuration file, the source database name is *mydb* and it has its logs on /mnt/dblogs and data on /mnt/dbdata. The clone database is to be renamed to *mydbcl*. It has its data on /mnt/dbdata_cl and logs on /mnt/dblogs_cl. The source database instance name is *db2inst1* and clone database instance name is *db2instc*.

Save the configuration file as /home/db2inst1/dbrelocate.cfg and execute the following command on the database server:

```
db2relocatedb -f /home/db2inst1/dbrelocate.cfg
```

3. Start the database manager instance you created for the migration test environment by executing the following command on the database server:

   ```
   db2start
   ```

   If the production database was offline during the Snapshot copy process, the clone database becomes available after executing **`db2start`**.

   The production database was online when the Snapshot copies were created. Therefore, use the **`db2inidb`** utility to initialize the clone database as a snapshot. You can do this by executing the following command on the database server:

   ```
   db2inidb database [DatabaseName] as snapshot
   ```

   Where `DatabaseName` identifies the name of the clone database that is going to be used for the test environment.

   For example, to initialize a clone database named *mydbcl*, execute the following commands on the database server:

   ```
   db2inidb mydbcl as snapshot
   ```

After completing these steps, the cloned database is ready. You can connect to the database and start using it.

If the source and the clone database need to be accessed from the same database host, then you must complete the additional steps described in Appendix B, “Configuring UNIX to access cloned and source databases in an SAN environment” on page 105.

In this scenario, the LUNs on the clone volume are accessed using FCP, but they can be accessed using the iSCSI access protocol as well.

5.5 Conclusions

The storage system offers the DB2 database administrator a compelling advantage and an elegant solution for database cloning. Use of the FlexVol clone feature combined with SnapMirror, enables database cloning at a remote storage system. The clone database can be used in multiple ways:

- To take load off the production database
- To use in the reporting environment
- To perform data mining
- To perform production-environment troubleshooting
- To refresh the test and development environment
- To work closely with live production data
- To enable easier and more convenient disaster recovery

Database cloning using N series technology is very easy and simple, sparing database administrators from working late hours and weekends just to replicate data for various IT operations. The storage space is used efficiently with Snapshot technology, which translates into savings in terms of dollars.

A

Configuring UNIX to access cloned and source databases in an NAS environment

In order to access the clone database and the source database from the same database server, complete the steps in this appendix.

A.1 Create a mount point for each clone volume

Execute the following command on the database server to create a mount point:

```
mkdir -p [MountPoint]
```

Where `MountPoint` identifies the name assigned to the mount location on the database server.

For example, to create a mount point named `/mnt/db2data_cl`, execute the following command on the database server:

```
mkdir -p /mnt/dbdata_cl
```

A.2 Define mount options

Define the mount options in the appropriate file for your operating system. For example, for Linux you define mount options in the /etc/fstab file. The mount should look similar to the following:

```
[StorageSystemName]:[FlexVolName] [MountPoint] nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

Where the following variables are defined as:

- `StorageSystemName` identifies the name assigned to the storage system that is used for the database storage.
- `FlexVolName` identifies the name assigned to the clone volume.
- `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, for a clone volume named *dbdata_cl* that resides on a IBM N Series storage system named *srcstore*, append the following mount entry to the /etc/fstab file on the database sever:

```
srcstore:dbdata_cl /mnt/dbdata_cl nfs
hard,rw,nointr,rsize=32768,wsize=32768,bg,vers=3,tcp 0 0
```

The mount entry should specify the IBM recommended mount options.

After appending the mount entry, you can mount the clone volume by executing the following command on the database server:

```
mount [MountPoint]
```

Where `MountPoint` identifies the name assigned to the mount location that is used to mount the flexible volume on the database server.

For example, to mount a clone volume that has mount entry specified in the /etc/fstab file, execute the following command on the second database server named *hostdst*:

```
mount /mnt/dbdata_cl
```

The database servers that we used had the Linux operating system.

A.3 Change ownership

In order to operate DB2 successfully, the DB2 instance owner should have ownership of the file systems on the clone volume that is mounted on the database server. Change the ownership by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where the following variables are defined as:

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the files system whose ownership is changed.

For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, execute the following command on the database server named *hostdst*:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

A.4 Rename the clone database

The clone database has the old database name and old tablespace container information. You must rename the clone database and update its tablespace header information to represent the new database and new tablespace containers. The **`db2relocatedb`** utility allows you to rename a database as well as to update the tablespace header information.

To rename the clone database:

1. Create a configuration file specifying both the new and old database names and tablespace containers. The configuration file should look similar to Example A-1.

Example: A-1 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
STORAGE_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/
```

2. Save the file as `dbrelocate.cfg` and grant execute permission on it. If the database was offline during the Snapshot copy process, then modify the database information by executing the **`db2relocatedb`** command:

   ```
   db2relocatedb -f [ConfigFile]
   ```

   Where `ConfigFile` identifies the name of the user-created configuration file based on that specifies new and old database information.

   For example, to update the tablespace containers information and rename the clone database using the sample configuration file named `/home/db2inst1/dbrelocate.cfg`, execute the following command on the database server:

   ```
   db2relocatedb -f /home/db2inst1/dbrelocate.cfg
   ```

   Note: When the **`db2inidb`** command with the relocate using option, is executed, it internally calls the **`db2relocatedb`** tool and can be used to relocate a database. As noted earlier, the **`db2inidb`** utility is used only to initialize a clone database that was created from an online database.

A.5 Check whether the database is cataloged

Check whether the database is cataloged correctly by executing the following command on the database server that is used to access the clone database:

```
db2 list db directory
```

The output from this command should look similar to that shown in Example A-2.

Example: A-2 Output from the db2 list command

```
System Database Directory
 Number of entries in the directory = 1

Database 1 entry:
 Database alias                       = MYPRD_CL
 Database name                        = MYPRD_CL
 Local database directory             = /mnt/dbdata_cl
 Database release level               = a.00
 Comment                              =
 Directory entry type                 = Indirect
 Catalog database partition number    = 0
```

A.6 The db2relocatedb command

Upon execution of the **`db2relocatedb`** command, the source database is automatically uncatalogged. You must recatalog the source database by executing the following command on the database server:

```
db2 "catalog database [DatabaseName] as [DatabaseAlias] on
[FileSystem]"
```

Where the following variables are defined as:

- `DatabaseName` identifies the name assigned to the database that is being cataloged.
- `DatabaseAlias` identifies the alias name assigned to the database that is being cataloged.
- `FileSystem` specifies the path on which the database being cataloged resides.

For example, to recatalog a source database named *mydb* that resides on the file system named /mnt/dbdata, execute the following command on the database server:

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

After completing the steps in this appendix, you should be able to access both the source and the cloned databases from the same database server.

B

Configuring UNIX to access cloned and source databases in an SAN environment

This appendix discusses how to access the clone database from the same database server.

B.1 List the LUN mappings

List the LUN mappings by executing the following command from the IBM N Series storage system:

```
lun show -m
```

The output from the **`lun show -m`** command should look similar to that shown in Example B-1.

Example: B-1 Output from lun show -m command

```
lun show -m
LUN path                Mapped to         LUN ID  Protocol
---------------------------------------------------------------
/vol/dbdata/data          host_src_fcp_igp      0     FCP
```

You can see from the output that the LUNs on the clone volume have the same mapping as the parent.

B.2 Remove the old mappings

Remove the old mapping for each LUN that exists on the clone volume by executing the following command on the storage system:

```
lun unmap [LunPath] [iGroupName]
```

Where the following variables are defined as:

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.

For example, to remove old mapping for a LUN named `/vol/dbdata_cl/data` from an igroup named `host_src_fcp_igp`, execute the following command on the storage system:

```
lun unmap /vol/dbdata_cl/data  host_src_fcp_igp
```

B.3 Create new mappings

Create a new mapping for the LUNs on the FlexClone volume by using a new ID and, by mapping them to a new igroup, or both. A LUN mapping can be created by executing the following command on the storage system:

```
lun unmap [LunPath] [iGroupName] [LunId]
```

Where the following variables are defined as

- `LunPath` identifies the name assigned to the new clone volume.
- `iGroupName` identifies the name assigned to the initiator group for the database host that is used to access the database.
- `LunId` identifies a numeric ID that is assigned to the LUN for mapping it to a specific initiator group.

For example, to map a LUN named `/vol/dbdata_cl/data` to an igroup named `host_dst_fcp_igp`, execute the following command on the storage system:

```
lun map /vol/dbdata_cl/data  host_dst_fcp_igp 0
```

B.4 Bring the clone online

After remapping, each LUN that is on the clone volume and that is used for the clone database must be brought online by executing the following command on the storage system:

```
lun online [LunPath]
```

Where `LunPath` identifies the name assigned to the new clone volume.

For example, to bring a LUN named `/vol/dbdata_cl/data` online, execute the following command on the storage system:

```
lun online /vol/dbdata_cl/data
```

B.5 Create Mount point

Create a mount point for each of the clone volumes by executing the following command on the database server:

```
mkdir -p [MountPoint]
```

Where `MountPoint` identifies the name assigned to the mount location on the database server.

For example, to create a mount point named `/mnt/db2data_cl`, execute the following command on the database server:

```
mkdir -p /mnt/dbdata_cl
```

B.6 Refresh the driver

Refresh the HBA driver on the database server. For example, to refresh a Qlogic FC HBA on a Linux database host, you would execute the following commands on the database server:

```
modprobe -r qla2300
modprobe -v qla2300
```

For any other operating system (OS) and HBA, refer to the OS reference manual and the HBA installation guide.

B.7 Obtain LUN device names

Obtain the LUN device names by executing the following command on the database server that is to be used to access the clone database:

```
sanlun lun show
```

B.8 Mount the LUN devices

Mount the LUN devices by executing the following command on the database server:

```
mount [DeviceName] [MountPoint]
```

Where the following variables are defined as:

- `DeviceName` identifies the name assigned to the new clone volume.
- `MountPoint` identifies the name assigned to the mount point that is used to mount the LUN device.

For example, to mount a LUN device that is identified by a name `/dev/sdb` by the database server, execute the following command on the database server:

```
mount /dev/sdb /mnt/dbdata
```

B.9 Change ownership

In order to operate DB2 successfully, the DB2 instance owner should have ownership of the LUN file systems mounted on the database server. Ownership can be changed by executing the following command on the database server:

```
chown -R [InstanceOwner]:[InstanceOwnerGroup] [FileSystem]
```

Where the following variables are defined as:

- `InstanceOwner` identifies the name assigned to the user who owns the database instance.
- `InstanceOwnerGroup` identifies the name assigned to the user's group that owns the database instance.
- `FileSystem` identifies the name of the file system whose ownership is being changed.

For example, to change ownership of the file system mounted on the mount point named `/mnt/dbdata_cl`, execute the following command on the database server named *hostdst*:

```
chown -R db2inst1:db2adm /mnt/dbdata_cl
```

B.10 Rename a cloned database

The cloned database has the old database name and old tablespace container information. You must rename the cloned database and update its tablespace header information to represent the new database and new tablespace containers. The **db2relocatedb** utility allows you to rename a database as well as update the tablespace header information. To rename a cloned database:

1. Create a configuration file specifying both new and old database names and tablespace containers. The configuration file should look similar to that shown in Example B-2.

Example: B-2 Configuration file

```
DB_NAME=mydb,mydbcl
DB_PATH=/mnt/dbdata,/mnt/dbdata_cl
INSTANCE=db2inst1
NODENUM=0
LOG_DIR=/mnt/dblogs/NODE0000,/mnt/dblogs_cl/NODE0000
CONT_PATH=/mnt/dbdata/*,/mnt/dbdata_cl/*
```

2. Save the file as dbrelocate.cfg, grant execute permission if necessary, and execute the db2relocatedb command as follows:

   ```
   db2relocatedb -f [ConfigFile]
   ```

   Where `ConfigFile` identifies the name assigned to the user-created configuration file that is used to relocate the database.

   For example, to update the tablespace containers information and rename the clone database using the sample configuration file named `/home/db2inst1/dbrelocate.cfg`, execute the following command on the database server:

   ```
   db2relocatedb -f /home/db2inst1/dbrelocate.cfg
   ```

B.11 Checking whether database is cataloged

Check whether the database is cataloged correctly by executing the following command on the database server:

```
db2 list db directory
```

The output from this command should look similar to that shown in Example B-3.

Example: B-3 Output from the db2 list command

```
db2 list db directory
 System Database Directory
 Number of entries in the directory = 1

Database 1 entry:
 Database alias                       = MYPRD_CL
 Database name                        = MYPRD_CL
 Local database directory             = /mnt/dbdata_cl
 Database release level               = a.00
Comment                              =
 Directory entry type                 = Indirect
Catalog database partition number    = 0
 Alternate server hostname            =
```

B.12 Recatalog the database

Upon execution of the **db2relocatedb** command, the source database is automatically uncatalogged. Recatalog the source DB2 database. Now you can access both source and cloned database from the same database server.

```
db2 "catalog database mydb as mydb on /mnt/dbdata"
```

After completing the steps in this appendix, you should be able to access both the source and the cloned databases from the same database server.

Related publications

We consider the publications that we list in this section to be particularly suitable for a more detailed discussion of the topics that we discuss in this IBM Redbook.

IBM Redbooks

For information about ordering these publications, see "How to get IBM Redbooks" on page 114. Note that some of the documents that we reference here might be available in softcopy only.

- *Using the IBM System Storage N Series with IBM Tivoli Storage Manager*, SG24-7243
- *IBM Storage System N series Antivirus Scanning Best Practices Guide*, REDP-4084
- *Setting up CIFS and Joining the Active Directory*, REDP-4074
- *Multiprotocol Data Access with IBM System Storage N series*, REDP-4176

Other publications

These publications are also relevant as further information sources:

- *IBM System Storage N3700 Installation and Setup Instructions,*GA32-0517-02
- *IBM System Storage N series N3700 Hardware and Service Guide,*GA32-0515-02
- *IBM System Storage N series Introduction and Planning Guide,* GA32-0543-01

Online resources

These Web sites are also relevant as further information sources:

- IBM NAS for your storage infrastructure

  http://www-03.ibm.com/servers/storage/nas/
- Support for System Storage™ N3700

  http://www-03.ibm.com/servers/storage/support/nas/n3700/
- Support for System Storage N5200

  http://www-03.ibm.com/servers/storage/support/nas/n5200/

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM Redbooks

Using IBM DB2 UDB with IBM System Storage N series

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Using IBM DB2 UDB with IBM System Storage N series

Redbooks

Exploiting FlexClone technology to build and use database clones

Creating a database on IBM N series systems

Using Snapshot and SnapMirror

Database cloning is the process by which you create an exact copy of a DB2 database, either by physically copying the data or by performing what is known as a *redirected restore*.

Database cloning is performed frequently by database administrators to provide near-production data for various business needs such as application development, QA testing, and report generation. Traditional methods of cloning a database pose various challenges, including system downtime and degraded system performance during the cloning process. Additionally, a large amount of storage space is required to store each clone. Furthermore, the maintenance overhead can be enormous if each cloned database requires a frequent data refresh.

This IBM Redbook describes the process used to create a clone of an IBM DB2 UDB database using FlexClone technology. This book also covers creating a database clone on a disaster recovery site that has replicated data using Data ONTAP SnapMirror technology.

SG24-7323-00 ISBN 0738496820