THE EXPERT'S VOICE® IN ORACLE

Oracle RMAN Database Duplication

FAST AND EASY PROVISIONING OF KNOWN-GOOD DATABASES TO DEVELOPERS AND TESTERS

Darl Kuhn

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

About IOUG Press

IOUG Press *is a joint effort by the* ***Independent Oracle Users Group (the IOUG)*** *and* ***Apress*** *to deliver some of the highest-quality content possible on Oracle Database and related topics. The IOUG is the world's leading, independent organization for professional users of Oracle products. Apress is a leading, independent technical publisher known for developing high-quality, no-fluff content for serious technology professionals. The IOUG and Apress have joined forces in IOUG Press to provide the best content and publishing opportunities to working professionals who use Oracle products.*

Our shared goals include:

- Developing content with excellence
- Helping working professionals to succeed
- Providing authoring and reviewing opportunities
- Networking and raising the profiles of authors and readers

To learn more about Apress, visit our website at **www.apress.com**. Follow the link for IOUG Press to see the great content that is now available on a wide range of topics that matter to those in Oracle's technology sphere.

Visit **www.ioug.org** to learn more about the Independent Oracle Users Group and its mission. Consider joining if you haven't already. Review the many benefits at www.ioug.org/join. Become a member. Get involved with peers. Boost your career.

www.ioug.org/join

Apress®

Introduction

Companies often have the requirement to provision copies of databases. These replicated databases are used for a wide variety of business purposes, such as reporting, testing Oracle upgrades, testing new application releases, troubleshooting, performance tuning, and so on. There are a wide variety of techniques for making copies of databases. Oracle provides many tools that can assist with database duplication. One such tool is the RMAN *DUPLICATE* functionality. With just a few lines of code DBAs can efficiently and easily create copies of databases.

The focus of this book is database duplication using RMAN. If you find yourself frequently replicating databases then consider using RMAN duplication to automate what would otherwise be a sometimes complicated process.

What This Book Covers

Chapter 1 introduces you to the topic of database duplication. The groundwork for the other chapters is provided here. Chapter 2 covers background information on manual techniques for making copies of databases. These methods are important to understand as they're widely used and will allow you to compare and contrast the manual methods with the RMAN duplication functionality.

Backup-based RMAN duplication is the focus of Chapter 3. This simplest way to duplicate a database is using an RMAN backup as the source. The targetless duplication feature is covered extensively here. Chapter 4 moves on to active duplication. This type of replication uses the live database as the source for duplicating a database. Chapter 5 then covers advanced topics such as duplicating subsets of tablespaces, enabling parallelism, container/pluggable database duplication, and RAC/ASM to non-RAC/non-ASM duplication.

Chapter 6 is an introduction to Oracle Net. Many of the duplication scenarios require you to connect to the source and destination databases over Oracle Net. This chapter serves as an introduction to Oracle Net implementation and management.

These six chapters will provide you with a solid foundation for Oracle RMAN database duplication. With this knowledge you'll be able to efficiently provision and replicate your company's databases; these skills are needed by all DBAs.

Conventions

The following typographical conventions are used in this book:

- $ is used to denote Linux/UNIX commands that can be run by the operating system owner of the Oracle binaries (usually named oracle).
- SQL> is used to denote one-line SQL*Plus statements.
- Monospaced font is used for code examples, utility names, file names, URLs, and directory paths.
- Italics are used to highlight a new concept or word.
- `UPPERCASE` indicates names of database objects like views, tables, and corresponding column names.
- < > is used where you need to provide input, such as a filename or password.

Source Code

The code for the examples shown in this book is available on the Apress web site, `www.apress.com`. A link can be found on the book's information page under the "Source Code/Downloads" tab. This tab is located underneath the Related Titles section of the page.

Errata

Apress makes every effort to make sure that there are no errors in the text or the code. However, to err is human, and as such we recognize the need to keep you informed of any mistakes as they're discovered and corrected. Errata sheets are available for all our books at `www.apress.com`. If you find an error that hasn't already been reported, please let us know. The Apress web site acts as a focus for other information and support, including the code from all Apress books, sample chapters, previews of forthcoming titles, and articles on related topics.

Contacting the Author

If you have any questions regarding the book, feel free to contact me directly at the following email address: `darl.kuhn@gmail.com`.

CHAPTER 1

Introduction

RMAN (Recovery Manager) is Oracle's flagship backup and recovery tool. Although RMAN's main purpose is the backup and recovery of Oracle databases, there are other features that make this utility invaluable to an organization. One salient feature of RMAN is that it allows you to efficiently copy a database from one location to another. The copy of the database can be placed on the same server as the source database or placed on a remote destination server. RMAN can use as its source either a live production database or an RMAN backup of a database. This is all effortlessly achieved through the RMAN *DUPLICATE* command. With just a few lines of code you can replicate a database from one environment to another. Furthermore this process is easily scripted for automation and repeatability.

I was a latecomer to using RMAN's duplication functionality. The genesis for my using this technology started a few years ago when I was assigned to work with a development team that required fairly frequent refreshes of development and testing environments from a copy of the production database. At first I grumbled about having to refresh environments for them. I'd tell them that there were several complicated steps involved in making a copy of a database. Indeed, for large databases it takes a great deal of time to copy data files and/or backups from server to server. Additional time is required to perform the restore and recovery of the copied database. I mandated that the team provide me at least a two-day warning when they required an environment to be replicated. The development manager then proposed that we duplicate various environments at a set time each week.

This is when I started using the *DUPLICATE* command. I figured out the *DUPLICATE* syntax required for each environment. I then encapsulated that logic in operating system shell scripts and additionally automated the running of these duplication jobs through an operating system scheduling utility (*cron*). Now each week this complex task of duplicating various environments has been reduced to processes automatically running and subsequent emails from the *cron* job informing me the duplication was successful.

I've also come across other scenarios wherein RMAN's duplicate functionality has vastly simplified what would otherwise be quite complex tasks. For instance, my manager asked me what it would take to replicate an Oracle real application cluster (RAC) database running on Oracle automatic storage management (ASM) to a non-RAC and non-ASM environment. He also asked if it would be possible to replicate the other way, to RAC/ASM from non-RAC/non-ASM. Given the complexity of RAC and ASM environments, this task at first seemed impossible. But the RMAN duplicate functionality turns it into a one-command operation.

Another situation I faced was replicating databases where the source database server was on a different operating system from the destination host; for example, having to duplicate from a Solaris-based system to a Linux server. Here again the RMAN duplicate functionality takes a seemingly difficult task and turns it into a few lines of code.

The purpose of this book is to provide clear and precise examples of how to take advantage of RMAN-based duplication in your work environment. Most examples will include a diagram that details the configuration for each scenario. The most commonly encountered problems for each situation are also explained.

This chapter starts by introducing the reasons for using RMAN duplication. Then I discuss the advantages and disadvantages of this feature. I also cover the basic setup that I use for the examples in this book. This information will form the basis for moving on to simple and then advanced RMAN duplication scenarios.

Use Cases for Duplicating

In many instances it's a requirement to replicate a database from one environment to another. There are many important reasons for this. Listed next are typical business requirements that drive the duplicating of a database.

- Offload reporting to a periodically refreshed copy of production
- Build development/test/quality assurance/beta environments
- Test database upgrades and migrations
- Test new application code
- Troubleshoot production issues
- Exercise backup and recovery strategy
- Create a Data Guard standby database used for disaster recovery and reporting

Database administrators (DBAs) add value by being able to quickly and effectively replicate a copy of one database to another. Manually duplicating a database is not difficult; however, it requires that the DBA perform about a dozen separate steps. This manual process can be time consuming and prone to error. RMAN vastly simplifies the database duplication operation by reducing it to a one-line command.

Methods for Replicating

Over the years, DBAs have devised many methods for transferring data from one database to another. Each method has its merits. Table 1-1 describes at a high level the most basic advantages and disadvantages of each replication technique. Not all possible replication tools are listed, nor are every single pro and con listed. Rather this table serves as a starting point for discussing when it is appropriate to use the RMAN duplicate functionality.

Table 1-1. *Replication techniques*

Technique	Advantages	Disadvantages
Copy of hot, cold, or RMAN backup	Most DBAs are familiar with these techniques, no extra license, fairly straightforward	Many manual steps involved, time consuming
Data Pump	DBAs are familiar with this tool, no extra license, excels at replicating specific schemas, tablespaces, and tables	Not as efficient when moving large amounts of data (e.g., hundreds of gigabytes)
Old exp/imp utilities	DBAs are familiar with these tools, easy to use	The *exp* utility is deprecated, Oracle strongly recommends to use Data Pump instead
Database link	Simple DML statements or Materialized Views to replicate from remote database	Not efficient for moving large amounts of data
External tables	Good for transporting data across different platforms and loading data from OS files	Many manual steps involved
GoldenGate	Oracle's flagship data-replication tool, multi-direction replication possible	Requires an extra license and maintenance of replication environment
SQL*Loader	Excellent for bulk-loading data from OS files, can load data over the network	Sometimes non-intuitive to craft SQL*Loader control file containing instructions
Pro*C or Java	Excellent for bulk-loading data from OS files	Requires knowledge of the language being used
Data Guard	Oracle's flagship disaster recovery tool, keeps standby database synchronized with primary database	Requires a DBA familiar with Data Guard, and for some advanced features requires an extra license
Enterprise Manager	Provides a graphical interface to many of the tools in this table	Requires an extra license, and some setup and maintenance is involved
Transportable tablespaces	Excellent for transporting data across different platforms	Some knowledge of Data Pump and/or RMAN is required, some downtime may be required for tablespaces being transported
Sync/Split technologies	Fast way to produce a copy of a large database	Costs money and requires maintenance
RMAN duplication	A few lines of code implements the replication of entire database, simple and efficient	Some knowledge of RMAN is required

As you can see, there are a wide variety of options available to replicate data. This table doesn't even contain every possible solution. For example, there are many third-party tools (outside of Oracle) that can be used to efficiently replicate databases. Rather, this table gives you an idea of what's possible and when to use potential solutions. Don't decide on a duplication option and then make the requirements fit it; instead, gather the replication requirements and determine which tool best meets those needs.

Even though I'm a huge fan of RMAN-based duplication, in the last year there have been some non-RMAN replication solutions that I've used; below I explain why:

- **External tables or SQL*Loader:** A customer sends a spreadsheet or csv file with a request to load it into the database. If I'm loading over a network link then I'll use SQL*Loader, otherwise I use external tables to load data from csv files.
- **exp/imp:** I got lazy once or twice and just wanted to export one table and so I decided to use the deprecated *exp* utility. Having said that, Data Pump is a vastly superior tool for any tasks that *exp*/*imp* have historically been used for.
- **Data Pump:** This tool really excels when you need to transport data at a granular level, such as by schema(s), by tablespace(s), or even by table(s). You can move schemas with or without the data, subsets of the data, subsets of tables within a schema, and so on. For instance, I use Data Pump quite often when just one table needs to be backed up, or when a schema and its objects are required to be replicated. This tool allows you to extract with a great deal of precision just the users or objects you want replicated.
- **Cold backup:** My manager asked me to make a copy of a small test database. The database wasn't in archive log mode, and there were no RMAN backups. In this situation I simply shutdown the test database, copied the data files to the remote server, then on the remote server re-created the control file, and lastly opened the database with *RESETLOGS*.

Having discussed various duplication techniques and reasons to use (or not) these methods, let's now focus on RMAN's duplication feature.

RMAN Duplicate Advantages

So when would you want to use the RMAN duplication functionality? RMAN excels when you need an entire database copied. The copy can be placed on the same server as the source database or on a remote server. RMAN seamlessly handles either requirement.

When you need a copy of a database that matches the source in every physical aspect, then RMAN is the tool to use. RMAN can efficiently duplicate very large databases across the network, between different operating system platforms, and from RAC/ASM to non-RAC/non-ASM (and vice versa). RMAN can use the active target database as its source, or it can use an RMAN backup for duplication. In these scenarios the RMAN *DUPLICATE* command has several advantages over other manual replication methods. I've divided these benefits into the three following subsections.

Ease of Use

The following aspects demonstrate the simplicity and flexibility of RMAN-based duplication:

- The *DUPLICATE* command is a standard part of RMAN (and therefore is a standard part of Oracle); there's no extra installation, license, or cost for this feature. RMAN ships with all editions of Oracle (although there are certain advanced features that are only available with the Enterprise Edition).
- Most Oracle DBAs are already familiar with RMAN; there's no extra training.
- The *DUPLICATE* command can be easily scripted for reliable repeatability and automation.
- You can duplicate from an RMAN backup of a database or directly from a live database.
- When duplicating, RMAN creates a new database identifier (DBID) for the duplicated database. This means you can register a duplicated database in the same recovery catalog as the target (source) database. Why is the new DBID important? Consider if you manually cloned a database (and didn't create a new DBID) and then tried to register the cloned database in the recovery catalog; this would cause confusion, as the recovery catalog will think the database already exists, thus throwing the *RMAN-20002: target database already registered…* error. This could cause your restore and recover mechanism to behave unexpectedly.
- You can create a duplicate database with a different name and different directory structure with a one-line command.
- Duplication to a user-specified point-in-time is available.

Performance and Security

This next list describes many of RMAN's default features that enhance performance and security when duplicating:

- You can allocate multiple channels to enable parallelization to maximize performance.
- As of Oracle 12c, RMAN by default uses the backup set format when replicating from an active database; this reduces resources required for active duplication and improves the performance.
- It's possible to duplicate encrypted databases.

- You can enable compression when duplicating active databases with the backup set format; this greatly reduces the network bandwidth required.
- RMAN provides options for duplication of a subset of tablespaces within a database in situations where you don't require the entire database to be replicated.

Flexible Replication

This list demonstrates that RMAN works seamlessly with other Oracle tools and across operating system platforms where Oracle is installed:

- RMAN is Data Guard–aware. You can use RMAN to create a standby database using the primary database as the source. This takes a somewhat manual and labor-DBA-intensive process and turns it into an automated one-line-of-code operation.
- You can replicate container and pluggable databases. It's also possible to duplicate subsets of pluggable databases that exist within a container database.
- It's possible to duplicate from non-RAC, non-ASM to RAC and ASM (and vice versa); this completely automates what otherwise would be a complex task.
- Some cross-platform duplication is supported. RMAN can assist in duplicating between servers using different operating systems.

The bottom line here is that RMAN's database duplication feature makes it easy to provision copies of databases. It's efficient, flexible, and simple to use. Every Oracle DBA should know how to proficiently use this tool.

RMAN Duplication Overview

Next I'll describe the basic terms involved and the duplication process. This information lays the foundation for understanding and leveraging the concepts in this book.

Definition of Terms

The following list provides a definition of RMAN duplication components. If you're not familiar with these terms, now would be a good time to review them.

- **Target (source) database.** This is the source database of the duplication operation. RMAN can duplicate directly from a live target database or a backup of the target database. The terms *target* and *source* are used interchangeably in this book.
- **Target (source) server.** The server that hosts the target database.

- **Auxiliary (destination) database.** This is the database copy that is created. This can exist on the same server as the target or on a different server. This is a complete stand-alone clone of the target database.
- **Auxiliary (destination) server.** The server that hosts the auxiliary database.
- **Active duplication.** RMAN uses the live target database data files as the source of the copy. No RMAN backup is necessary for active duplication.
- **Backup set format.** When using active duplication, RMAN replicates the data files across the network in a backup set format. This is the same format as if it were copying from an RMAN backup set. No backup set is created on disk; rather, it's the format RMAN uses to transfer the data files across the network. This format is used by default and is available with Oracle 12c and above.
- **Image copy format.** When using active duplication you can specify that RMAN use image copies. Image copies are byte-for-byte copies of the data files. This is the only format available in active-based duplication in Oracle 11g.
- **Backup-based duplication.** RMAN uses a backup of the target database to create the auxiliary database.
- **Targetless duplication.** A form of backup-based duplication. No connection to the target database (or recovery catalog) is required.
- **Standby database.** A near real-time copy of a target database kept in synchronization with the target database via Oracle's Data Guard tool. The *DUPLICATE* command has an option to specify that a standby database be created for the target (primary) database.
- **Recovery catalog.** An optional storage repository of the target database RMAN metadata. This same metadata is stored in the target database control file.
- **Channels.** The Oracle server processes for handling I/O between files being copied and restored and the backup device (disk or tape).

Now that we've reviewed the basic duplication components and definitions, we'll discuss the detailed steps that RMAN performs when duplicating.

RMAN Duplication Process

Understanding the RMAN duplication process sequence of events lays the foundation for successfully using this technology. At a high level, when you issue a *DUPLICATE* command, RMAN first allocates at least one channel (more if you specify them). RMAN uses this channel to restore the target database control file to the auxiliary server. Next, the auxiliary database is mounted using the control file that was just restored. RMAN then restores to the auxiliary server the data files, incremental backups (if any exist and are required), and archive redo logs. RMAN then recovers the auxiliary database (applies increment backups and/or archive redo logs). Lastly, it sets a new DBID and opens the database with the *RESETLOGS* option. This process is depicted in Figure 1-1.

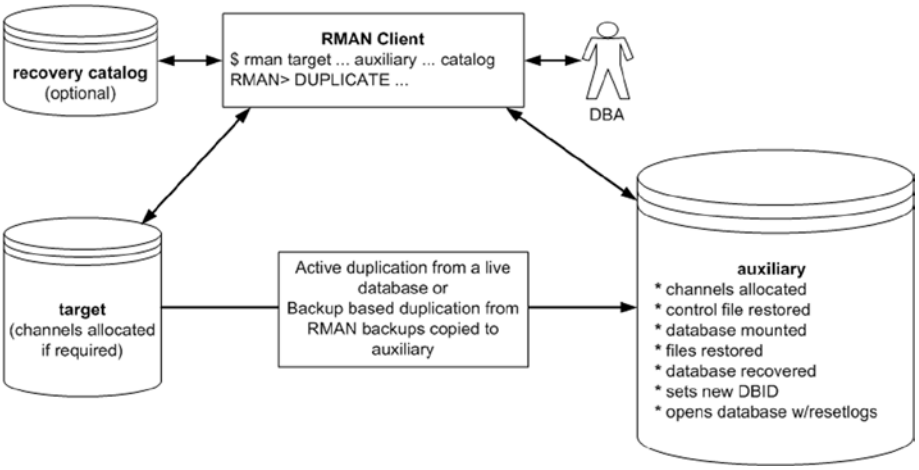

Figure 1-1. *High-level view of RMAN duplication*

Now that you have a bird's eye view of the process and components involved, let's look at the detailed steps RMAN performs when duplicating a database.

1. DBA accesses RMAN client and connects to auxiliary database. If required, the DBA will also establish connections to the target and/or recovery catalog.
2. DBA issues RMAN *DUPLICATE* command.
3. RMAN allocates channels on the auxiliary database and/or target database as required.
4. RMAN creates a default server parameter file (SPFILE) for the auxiliary database if not duplicating to a standby database and not duplicating the SPFILE, and if the auxiliary instance was not started with an SPFILE.

5. RMAN restores from backup sets or image copies from the target database the latest control file that satisfies the *UNTIL* clause requirements. If no *UNTIL* clause was specified, then RMAN uses the maximum system change number (SCN) in the last archived redo log generated by the target database as recorded in *V$ARCHIVED_LOG*.

■ **Note** The SCN is Oracle's internal clock or counter. The SCN is incremented with every change within the database. This information is critical for read consistency and recovery operations; it provides the exact sequence in which changes occurred in the database.

---

6. Opens the auxiliary database in mount mode using the control file that was restored in the prior step.
7. Determines from the RMAN repository (either from a restored control file, an active connection to the target database, or a connection to the recovery catalog) which backups will be used for restoring the data files to the auxiliary database.
8. Restores the target database data files to the auxiliary database and applies any applicable incremental backups and/or archived redo log files. If no *UNTIL* clause is specified, RMAN will apply the last archive redo log that was restored. For active duplication the target database data files are the source. For targetless duplication the DBA places the RMAN target database backups where the auxiliary server can access them (e.g., copies the backups to the auxiliary server).
9. Shuts down the auxiliary database.
10. Restarts the auxiliary database in nomount mode.
11. Creates a new control file by which a new DBID is created for the auxiliary database and stores the new DBID in the restored data files.
12. Opens the auxiliary database with the *RESETLOGS* clause, which creates new online redo logs for the auxiliary database.

This list of steps provides a basic understanding of RMAN's duplication process. When working through issues refer back to this list and logically think through the steps RMAN is performing and how that applies to the issue you're trying to resolve. You'll see in coming chapters that the RMAN *DUPLICATE* command produces a vast amount of output. When you have issues, knowing the sequence of steps RMAN performs can help you pinpoint at what stage the issue is occurring and thus assist in resolving the problem.

■ **Tip** Starting with Oracle 12c, if you do not want the duplication process to open the database (as the final step), use the *NOOPEN* clause of the *DUPLICATE* command.

Example Setup Environment

This section provides details regarding the environment that I'll use throughout the book in most of the examples. For most duplication scenarios I use two servers. The target database server is named *shrek* and the auxiliary server is named *shrek2*. The target database is named *TRG*. The auxiliary database is either named *TRG* (when there's no database name change) or *DUP* (when the database is renamed during duplication). The target server data files are located in */u01/dbfile/TRG*. The auxiliary server data files are restored to */u01/dbfile/TRG* when the directory name is the same as the target. If the directory structure is different, then the directory name is */u01/dbfile/DUP*. This environment is displayed in Figure 1-2.

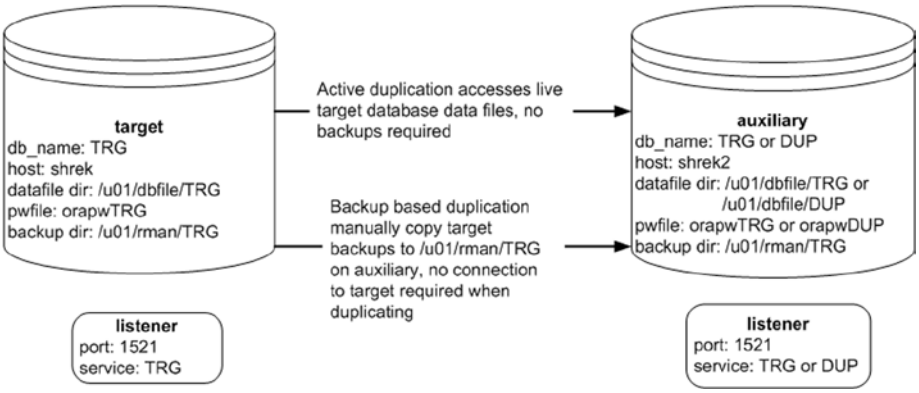

Figure 1-2. *Basic configuration for RMAN duplication examples*

Additionally, in this configuration the same exact same version of the Oracle software (binaries) is installed on both servers (*shrek* and *shrek2*). RMAN will not let you connect to both a target database and an auxiliary database of different Oracle database versions. If you attempt to do that you may receive a message similar to the following:

```
RMAN-06618: RMAN client and database release mismatch...
```

Also, for active duplication there must be Oracle Net connectivity between the servers. For targetless duplication no listener or connection to the target database is required. Targetless duplication only requires access to an RMAN backup of the target database. The RMAN backup can be placed on shared storage readable from the auxiliary server or it can be directly copied to the auxiliary server. For the targetless duplication examples in this book I directly copy the RMAN backup to the auxiliary server.

■ **Note** Any time you connect over Oracle Net to a database as a *SYSDBA* privileged user (such as *SYS*) or with user-assigned *SYSBACKUP* privileges, you must have a password file in place on the remote server.

Summary

Each Oracle installation includes a ready-to-go RMAN executable. RMAN is chiefly used for backup and recovery operations. However, RMAN can also be used to replicate database environments. RMAN's *DUPLICATE* command is a very powerful feature used to make a copy of a database. The replicated database is a stand-alone copy of the source (target) database. The duplicated database can be used for a variety of business requirements, such as offloading reporting, testing, troubleshooting, and creating a Data Guard standby database.

The duplication process is reliable and simple. RMAN provides many built-in features to achieve performance and security. RMAN is capable of duplicating in complex database environments such as container and associated pluggable databases, RAC and ASM, and differing operating systems.

This chapter serves as an introduction the RMAN duplication functionality. The next chapter in this book walks you through various manual replication processes. This information is relevant because it helps you understand the ease of using RMAN duplication to achieve the same goals. A knowledge of manual methods also assists in troubleshooting activities. You'll be better prepared to face any type of duplication situation if you're knowledgeable about manual procedures. If you're already familiar with manual replication methods then feel free to move on to Chapters 3, 4, and 5, where the bulk RMAN duplication is investigated and explained.

CHAPTER 2

Manual Duplication Techniques

This chapter will walk you through manual methods for cloning databases and tablespaces. If you're already familiar with these techniques, then feel free to move on to the next chapters in this book, which illustrate how to employ the RMAN duplication process. Otherwise, there are several manual replication scenarios covered in this chapter:

- Cold backup
- RMAN backup
- Data Pump across network link
- Data Pump and transportable tablespaces
- RMAN and transportable tablespaces
- External tables

Knowledge of these methods will help you understand when it's appropriate to use a technique and its advantages and disadvantages. This information will help you better understand the other chapters in this book that contrast these techniques with the RMAN *DUPLICATE* functionality (covered in Chapters 3, 4, and 5). First up is cloning a database using a cold backup.

Cloning from Cold Backup

If you worked with Oracle twenty or so years ago, you probably used a cold backup to move a database from one server to another. Even though this is an old technique, I still find myself occasionally using this method for cloning a database. For example, recently my supervisor asked me to copy a database (about 20 gig in size) from one server to another. In this scenario, the destination server directory structure was different from the source server directory structure, and the destination database

needed to have a different name that the source database. In this situation I used a cold backup to move the database for the following reasons:

- Source database wasn't in archive log mode
- There weren't any RMAN backups of the source database
- Source database was fairly small, and it wouldn't take long to copy the data files
- The requirement was to replicate the entire database

For this example to work you need the same version of Oracle installed on both the source and destination servers. The scenario is depicted in Figure 2-1.

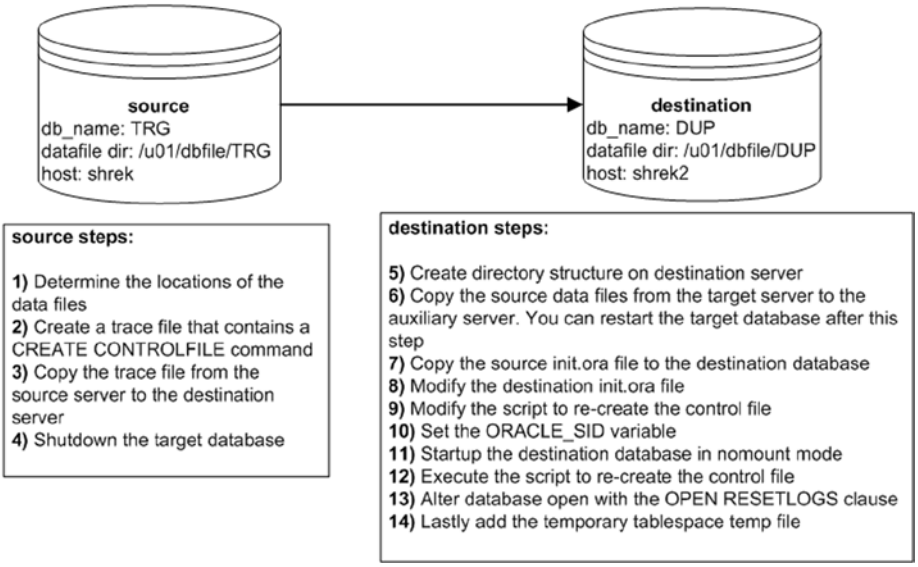

Figure 2-1. *Cloning with a cold backup*

Next are the detailed descriptions of each of the steps shown in Figure 2-1.

1. On the source database, determine the locations of the data files:

```
SQL> select name from v$datafile;
```

Here's some output for the database used in this example:

```
NAME
--------------------------------
/u01/dbfile/TRG/repdata.dbf
/u01/dbfile/TRG/repidx.dbf
/u01/dbfile/TRG/sysaux01.dbf
/u01/dbfile/TRG/system01.dbf
/u01/dbfile/TRG/undotbs01.dbf
/u01/dbfile/TRG/users01.dbf
```

2. On the source database create a trace file that contains a *CREATE CONTROLFILE* command in it:

```
SQL> alter database backup controlfile to trace as '/tmp/dk.sql' resetlogs;
```

3. Copy the trace file from the source server to the destination server. This example uses the Linux/UNIX *scp* command (initiated from the source server):

```
$ scp /tmp/dk.sql oracle@shrek2:/tmp
```

4. Shut down the source database using *immediate* (and not *abort*):

```
$ sqlplus / as sysdba
SQL> shutdown immediate;
```

5. Create directory structures on destination server:

```
$ mkdir /u01/dbfile/DUP
$ mkdir /u01/oraredo/DUP
```

6. While the source database is shut down, copy the source data files from the source server to the destination server. This example uses the Linux/UNIX *scp* command (initiated from the destination server):

```
$ scp oracle@shrek:/u01/dbfile/TRG/*.dbf /u01/dbfile/DUP
```

Notice there's no need to copy the control files or the online redo logs in this scenario. Since the destination directory structure and destination database name will be different from the source name, the control files and online redo logs will need to be recreated. If the directory structure and the database name were the same on both the source and the destination, the procedure would be as simple as shutting down the source database, copying all control files, data files, online redo logs, and initialization file to the destination server, and then starting the database.

■ **Note** After the copy is complete you can restart the source database.

7. Copy the source *init.ora* file to the destination server. If your source database uses an SPFILE, then you can create a text-based *init.ora* file from SQL*Plus, as follows:

```
SQL> create pfile from spfile;
```

This command will place a text-based initialization file with the name of *init<SID>.ora* in the *ORACLE_HOME/dbs* directory. If you don't want the text-based file to be placed in that directory you can override the default behavior as follows:

```
SQL> create pfile='/tmp/initTRG.ora' from spfile;
```

This example uses the Linux/UNIX *scp* command (initiated from the destination server). Modify this appropriately for your environment. Assuming the file is in the default location of *ORACLE_HOME/dbs*:

```
$ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora $ORACLE_HOME/dbs/initDUP.ora
```

8. Modify the destination *init.ora* file, change the *DB_NAME* parameter to reflect the new name of the database, and also modify any directories to reflect the directory structure of the destination environment:

```
$ vi $ORACLE_HOME/dbs/initDUP.ora
```

Here is the content of the *initDUP.ora* file after the modifications:

```
db_name='DUP'
control_files='/u01/dbfile/DUP/control01.ctl','/u01/dbfile/DUP/control02.ctl'
db_block_size=8192
fast_start_mttr_target=500
job_queue_processes=10
memory_max_target=500M
memory_target=500M
open_cursors=100
os_authent_prefix=''
processes=100
remote_login_passwordfile='EXCLUSIVE'
resource_limit=true
undo_management='AUTO'
undo_tablespace='UNDOTBS1'
workarea_size_policy='AUTO'
```

9. Modify the script to recreate the destination database control file:

```
$ vi /tmp/dk.sql
```

Change the first line to include the *SET* keyword and change the database name and directory structures to reflect the destination environment. Here are the contents of *dk.sql* after the modifications:

```
CREATE CONTROLFILE REUSE SET DATABASE "DUP" RESETLOGS ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 4
    MAXDATAFILES 1024
    MAXINSTANCES 1
    MAXLOGHISTORY 876
LOGFILE
  GROUP 1 '/u01/oraredo/DUP/redo01a.rdo'  SIZE 50M BLOCKSIZE 512,
  GROUP 2 '/u01/oraredo/DUP/redo02a.rdo'  SIZE 50M BLOCKSIZE 512
DATAFILE
  '/u01/dbfile/DUP/system01.dbf',
  '/u01/dbfile/DUP/sysaux01.dbf',
  '/u01/dbfile/DUP/undotbs01.dbf',
  '/u01/dbfile/DUP/users01.dbf',
  '/u01/dbfile/DUP/repdata.dbf',
  '/u01/dbfile/DUP/repidx.dbf'
CHARACTER SET AL32UTF8;
```

10. Set the *ORACLE_SID* variable to reflect the destination database name:

```
$ export ORACLE_SID=DUP
```

11. Start up the destination database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
```

12. Execute the script to recreate the control file:

```
SQL> @/tmp/dk.sql
```

You should see this message if successful:

```
Control file created.
```

At this point you have new control files and the database is in mount mode.

13. Alter the destination database open with the *OPEN RESETLOGS* clause:

```
SQL> alter database open resetlogs;
Database altered.
```

14. Lastly, add the temporary tablespace temp file:

```
SQL> ALTER TABLESPACE TEMP ADD TEMPFILE '/u01/dbfile/DUP/temp01.dbf'
     SIZE 524288000 REUSE AUTOEXTEND OFF;
```

Also keep in mind that other steps may be required for your environment depending on your standards. For example, you might want to ensure the database SID is listed in the *oratab* file, or you might require the use of an SPFILE, enabling a password file, changing passwords, enabling archiving, taking a backup, adding entries into Oracle Net files, and so on.

The advantages of the cold backup approach to cloning a database are:

- It's fairly simple and not much can go wrong (which simplifies troubleshooting any issues). There aren't many moving parts to this technique.
- It uses a combination of SQL and operating system commands, so you don't need to be familiar with any other tools to accomplish this task. A savvy manager, system administrator, or developer could easily use this approach to replicate a database.

The downside to this approach is that it requires you to shut down the source database while it is being copied. Thus, if you work in an environment that can't afford any downtime with the source database, then this approach isn't appropriate.

Copying from an RMAN Backup

When you think about architecting your backup strategy, as part of the process you must also consider how you're going to restore and recover. Your backups are only as good as the last time you tested a restore and recovery. A backup can be rendered worthless without a good restore and recovery strategy. The last thing you want to happen is to experience a media failure, go to restore your database, and then find out you're missing a file, you don't have enough space to restore, something is corrupt, and so on.

One of the best ways to test an RMAN backup is to restore and recover it to a different database server. This will exercise all your backup, restore, and recovery DBA skills. If you can restore and recover an RMAN backup on a different server, it will give you confidence when a real disaster hits. Moving a database from one server to another using an RMAN backup requires an expert-level understanding of the Oracle architecture and how backup and recovery works. The next example will do just that; it uses an RMAN backup to restore and recover a database on a different server. This scenario is depicted in Figure 2-2.

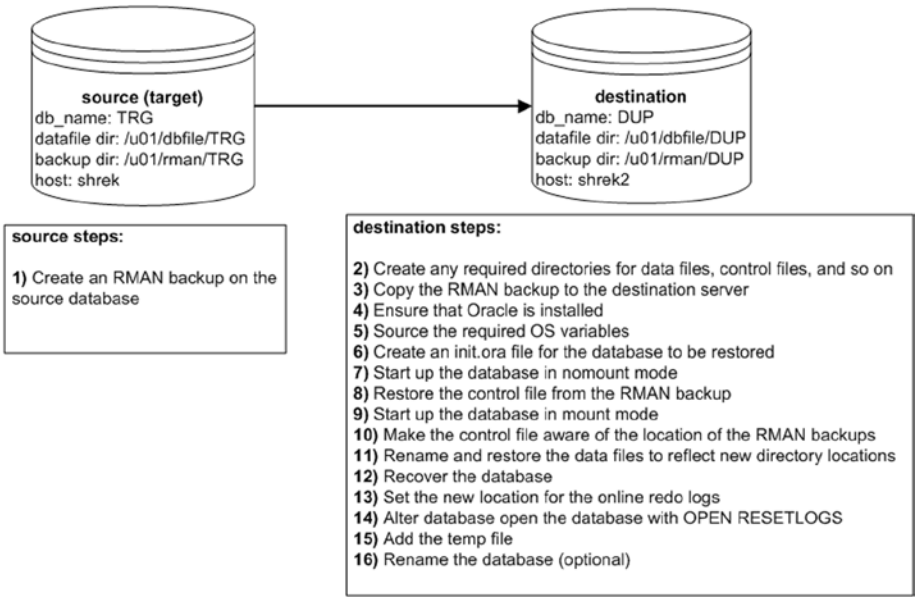

Figure 2-2. *Manually cloning a database using an RMAN backup*

Notice in Figure 2-2 that only step 1 occurs on the source database server. All remaining steps are performed on the destination server. For this example the source database is named *TRG*, and the destination database is named *DUP*. Also notice that the originating source server and destination server have different directory names. You'll have to adjust these directory names to reflect the directory structures on your database servers. Let's get started with step 1:

1. Create an RMAN backup on the source (target) database. When backing up a database, make sure you have the autobackup control file feature turned on:

```
$ rman target /
RMAN> configure controlfile autobackup on;
```

Also include the archive redo logs as part of the backup, as shown:

```
RMAN> backup database plus archivelog;
```

Verify that a backup of the control file exists:

```
RMAN> list backup of controlfile;
```

Here's some sample output:

```
Piece Name: /u01/rman/TRG/TRGctl_c-1251088236-20141228-00.bk
```

You'll need to reference the prior backup piece file when you restore the control file on the destination server (step 8). Also notice for this example that the backup pieces on the source server are in the */u01/rman/TRG* directory.

2. On the destination server, create any required directories for data files, control files, and so on. For this example the destination server directories created are:

```
$ mkdir -p /u01/rman/DUP
$ mkdir -p /u01/dbfile/DUP
$ mkdir -p /u01/oraredo/DUP
$ mkdir -p /u01/arch/DUP
```

3. Copy the RMAN backup to the destination server. This exampl.nux/UNIX *scp* command to copy the backup pieces (initiated from the destination server):

```
$ scp oracle@shrek:/u01/rman/TRG/*.*  /u01/rman/DUP
```

■ **Note** If the RMAN backups are on tape instead of on disk, then the same media manager software must be installed/configured on the destination server. Also, that server must have direct access to the RMAN backups on tape.

---

4. On the destination server, ensure you have the same version of the Oracle binaries installed as you do on the originating database.

5. On the destination server establish the OS variables, such as *ORACLE_SID*, *ORACLE_HOME*, and *PATH*. The *ORACLE_SID* variable is initially set to match what it was on the source database (*TRG* in this example). The destination database name will be changed as part of the last step in this list, to *DUP*. Here are the settings for *ORACLE_SID* and *ORACLE_HOME* on the destination server:

```
$ export ORACLE_SID=TRG
$ echo $ORACLE_SID
TRG
$ echo $ORACLE_HOME
/orahome/app/oracle/product/12.1.0.2/db_1
```

6. Copy the *init.ora* file from the source server to the destination server, placing it in the *ORACLE_HOME/dbs* directory. Modify the *init.ora* file so that it matches the destination box in terms of any directory paths. Ensure that you change the parameters, such as the *CONTROL_FILES*, to reflect the new path directories on the destination server (*/u01/dbfile/DUP*, in this example). Initially, the name of the *init.ora* file on the destination server is *initTRG.ora*, and the name of the database is *TRG*. Both will be renamed in a later step. Here are the contents of the *initTRG.ora* file:

```
db_name='TRG'
control_files='/u01/dbfile/DUP/control01.ctl','/u01/dbfile/DUP/control02.ctl'
log_archive_dest_1='LOCATION=/u01/arch/DUP'
log_archive_format='DUP%t_%s_%r.arc'
db_block_size=8192
fast_start_mttr_target=500
job_queue_processes=10
memory_max_target=800M
memory_target=800M
open_cursors=100
processes=100
remote_login_passwordfile='EXCLUSIVE'
resource_limit=true
standby_file_management='auto'
undo_management='AUTO'
undo_tablespace='UNDOTBS1'
workarea_size_policy='AUTO'
```

7. You should now be able to start up the destination database in nomount mode:

```
$ rman target /
RMAN> startup nomount;
```

8. Next, restore the control file from the backup that was previously copied from the source database; for example:

```
RMAN> restore controlfile from
'/u01/rman/DUP/TRGctl_c-1251088236-20141228-00.bk';
```

The control file will be restored to all locations specified by the *CONTROL_FILES* initialization parameter in the destination *init.ora* file. Here is some sample output from the restore operation:

```
channel ORA_DISK_1: restoring control file
channel ORA_DISK_1: restore complete, elapsed time: 00:00:07
output file name=/u01/dbfile/DUP/control01.ctl
output file name=/u01/dbfile/DUP/control02.ctl
```

You may see an error like this:

```
RMAN-06172: no AUTOBACKUP found or specified handle ...
```

In this situation, ensure that the path and backup piece names are correctly specified.

9. You should now be able to start up your database in mount mode:

```
RMAN> alter database mount;
```

At this point, your control files exist and have been opened, but none of the data files or online redo logs exist yet.

10. Make sure the control file is aware of the location of the RMAN backups. First, use the *CROSSCHECK* command to let the control file know that none of the backups or archive redo logs are in the same location that they were in on the original server:

```
RMAN> crosscheck backup; # Crosscheck backups
RMAN> crosscheck copy;   # Crosscheck image copies and archive logs
```

You'll probably see output indicating that RMAN can't validate that archive redo logs exist:

```
archived log file name=/u01/arch/TRG/TRG1_16_869840124.arc
RECID=765 STAMP=869842623
```

That's the expected behavior because those archive redo logs do not exist on the destination server.

Next use the *CATALOG* command to make the control file aware of the location and names of the backup pieces that were copied to the destination server.

■ **Note** Don't confuse the *CATALOG* command with the recovery catalog schema. The *CATALOG* command adds RMAN metadata to the control file, whereas the recovery catalog schema is a user, generally created in a separate database, which can be used to store RMAN metadata.

In this example, any RMAN files that are in the */u01/rman/DUP* directory will be cataloged in the control file:

```
RMAN> catalog start with '/u01/rman/DUP';
```

Here is some sample output:

```
List of Files Unknown to the Database
=====================================
File Name: /u01/rman/DUP/TRGctl_c-1251088236-20141228-00.bk
File Name: /u01/rman/DUP/TRGrman1_b7pr9m9q_1_1.bk
File Name: /u01/rman/DUP/TRGrman2_b6pr9m82_1_1.bk
File Name: /u01/rman/DUP/TRGrman2_b4pr9m6k_1_1.bk
File Name: /u01/rman/DUP/TRGrman1_b2pr9m4c_1_1.bk
File Name: /u01/rman/DUP/TRGrman2_b3pr9m4c_1_1.bk
File Name: /u01/rman/DUP/TRGrman1_b5pr9m82_1_1.bk
Do you really want to catalog the above files (enter YES or NO)?
```

Now, type *YES* (if everything looks okay). You should then be able to use the RMAN *LIST BACKUP* command to view the newly cataloged backup pieces:

```
RMAN> list backup;
```

You should see output indicating that RMAN is aware of the backups that were copied to the destination server. Here's a small snippet of the output:

```
BP Key: 280   Status: AVAILABLE  Compressed: NO  Tag:
TAG20150108T203552
Piece Name: /u01/rman/DUP/TRGrman2_jkps7th9_1_1.bk
```

11. Rename and restore the data files to reflect new directory locations. If your destination server has the exact same directory structure as the original server directories, you can issue the *RESTORE* command directly:

```
RMAN> restore database;
```

However, when restoring data files to locations that are different from the original directories, you'll have to use the *SET NEWNAME* command. Create a file that uses an RMAN *run{}* block that contains the appropriate *SET NEWNAME* and *RESTORE* commands. I like to use a SQL script that generates SQL to give me a starting point. Here is a sample script:

```
set head off feed off verify off echo off pages 0 trimspool on
set lines 132 pagesize 0
spo newname.sql
--
```

```
select 'run{' from dual;
--
select
'set newname for datafile ' || file# || ' to ' || '''' || name || '''' || ';'
from v$datafile;
--
select
'restore database;' || chr(10) ||
'switch datafile all;' || chr(10) ||
'}'
from dual;
--
spo off;
```

Run the prior script from SQL*Plus as *SYS*. In this example, the prior code is placed in a file named *gen.sql* and executed as follows:

```
SQL> @gen.sql
```

After running the script, these are the contents of the *newname.sql* script that was generated:

```
run{
set newname for datafile 1 to '/u01/dbfile/TRG/system01.dbf';
set newname for datafile 2 to '/u01/dbfile/TRG/sysaux01.dbf';
set newname for datafile 3 to '/u01/dbfile/TRG/undotbs01.dbf';
set newname for datafile 4 to '/u01/dbfile/TRG/users01.dbf';
set newname for datafile 5 to '/u01/dbfile/TRG/repdata.dbf';
set newname for datafile 6 to '/u01/dbfile/TRG/repidx.dbf';
restore database;
switch datafile all;
}
```

Then, modify the contents of the *newname.sql* script to reflect the directories on the destination database server. Here is what the final *newname.sql* script looks like for this example:

```
run{
set newname for datafile 1 to '/u01/dbfile/DUP/system01.dbf';
set newname for datafile 2 to '/u01/dbfile/DUP/sysaux01.dbf';
set newname for datafile 3 to '/u01/dbfile/DUP/undotbs01.dbf';
set newname for datafile 4 to '/u01/dbfile/DUP/users01.dbf';
set newname for datafile 5 to '/u01/dbfile/DUP/repdata.dbf';
set newname for datafile 6 to '/u01/dbfile/DUP/repidx.dbf';
restore database;
switch datafile all;
}
```

Now, connect to RMAN and run the prior script to restore the data files to the new locations:

```
$ rman target /
RMAN> @newname.sql
```

Here's a small sample of the output from the prior script:

```
executing command: SET NEWNAME
executing command: SET NEWNAME
...
channel ORA_DISK_1: restoring datafile 00001 to /u01/dbfile/DUP/system01.dbf
channel ORA_DISK_1: restoring datafile 00004 to /u01/dbfile/DUP/users01.dbf
...
input datafile copy RECID=16 STAMP=869854446 file
name=/u01/dbfile/DUP/repidx.dbf
RMAN> **end-of-file**
```

All the data files have been restored to the new database server. You can use the RMAN *REPORT SCHEMA* command to verify that the files have been restored and are in the correct locations:

```
RMAN> report schema;
```

Here is some sample output:

```
RMAN-06139: WARNING: control file is not current for REPORT SCHEMA
Report of database schema for database with db_unique_name TRG
List of Permanent Datafiles
===========================
File Size(MB) Tablespace           RB segs Datafile Name
---- -------- -------------------- ------- ------------------------
1    500      SYSTEM               ***     /u01/dbfile/DUP/system01.dbf
2    500      SYSAUX               ***     /u01/dbfile/DUP/sysaux01.dbf
3    200      UNDOTBS1             ***     /u01/dbfile/DUP/undotbs01.dbf
4    10       USERS                ***     /u01/dbfile/DUP/users01.dbf
5    10       REPDATA              ***     /u01/dbfile/DUP/repdata.dbf
6    10       REPIDX               ***     /u01/dbfile/DUP/repidx.dbf
List of Temporary Files
=======================
File Size(MB) Tablespace           Maxsize(MB) Tempfile Name
---- -------- -------------------- ----------- --------------------
1    500      TEMP                 500         /u01/dbfile/TRG/temp01.dbf
```

From the prior output you can see that the database name and temporary tablespace data file still don't reflect the destination database (*DUP*). These will be modified in subsequent steps.

12. Next you need to apply any archive redo files that were generated during the backup. These should be included in the backup because the *ARCHIVELOG ALL* clause was used to create the backup. Initiate the application of redo files via the *RECOVER DATABASE* command:

```
RMAN> recover database;
```

RMAN will restore and apply as many archive redo logs as it has in the backup pieces; it may throw an error when it reaches an archive redo log that doesn't exist. For example:

```
RMAN-06054: media recovery requesting unknown archived log for...
```

That error message is fine. The recovery process will restore and recover archive redo logs contained in the backups, which should be sufficient to open the database. The recovery process doesn't know when to stop applying archive redo logs and therefore will continue to attempt to do so until it can't find the next log. Having said that, now is a good time to verify that your data files are online and not in a fuzzy state:

```
SQL> select file#, status, fuzzy, error, checkpoint_change#,
to_char(checkpoint_time,'dd-mon-rrrr hh24:mi:ss') as checkpoint_time
from v$datafile_header;
```

Here is a small sample of the output:

```
     FILE# STATUS  FUZ ERROR      CHECKPOINT_CHANGE# CHECKPOINT_TIME
---------- ------- --- ---------- ------------------ --------------------
         1 ONLINE  NO                     1.3790E+13 23-jan-2015 15:23:37
         2 ONLINE  NO                     1.3790E+13 23-jan-2015 15:23:37
...
```

If you do have a file with a fuzzy status of *YES*, this indicates more redo logs need to be applied to the data file (normally this should not happen in this scenario).

13. Set the new location for the online redo logs. If your source and destination servers have the exact same directory structures, then you don't need to set a new location for the online redo logs (so you can skip this step). However, if the directory structures are different, then you'll need to update the control file to reflect the new directory for the online redo logs. I sometimes use an SQL script that generates SQL to assist with this step:

```
set head off feed off verify off echo off pages 0 trimspool on
set lines 132 pagesize 0
spo renlog.sql
select
```

```
'alter database rename file ' || chr(10)
|| '''' || member || '''' || ' to ' || chr(10) || '''' || member || ''''
||';'
from v$logfile;
spo off;
set feed on verify on echo on
```

For this example, assume the prior code was placed in a file named *genredo.sql* and run it as follows:

```
SQL> @genredo.sql
```

Here is a snippet of the *renlog.sql* file that was generated:

```
alter database rename file
'/u01/oraredo/TRG/redo01a.rdo' to
'/u01/oraredo/TRG/redo01a.rdo';

alter database rename file
'/u01/oraredo/TRG/redo02a.rdo' to
'/u01/oraredo/TRG/redo02a.rdo';
```

The contents of *renlog.sql* need to be modified to reflect the directory structure on the destination server. Here is what *renlog.sql* looks like after being edited:

```
alter database rename file
'/u01/oraredo/TRG/redo01a.rdo' to
'/u01/oraredo/DUP/redo01a.rdo';

alter database rename file
'/u01/oraredo/TRG/redo02a.rdo' to
'/u01/oraredo/DUP/redo02a.rdo';
```

Update the control file by running the *renlog.sql* script:

```
SQL> @renlog.sql
```

You can select from *V$LOGFILE* to verify that the online redo log names are correct:

```
SQL> select member from v$logfile;
```

Here is the output for this example:

```
/u01/oraredo/DUP/redo01a.rdo
/u01/oraredo/DUP/redo02a.rdo
```

14. You must open the database with the *OPEN RESETLOGS* clause (because there are no online redo logs, and they must be recreated at this point):

```
SQL> alter database open resetlogs;
```

If successful, you should see this message:

```
Statement processed
```

■ **Note** Keep in mind that all the passwords from the newly restored copy are as they were in the source database. You may want to change the passwords in a replicated database, especially if it was copied from production.

---

15. Add the temporary tablespace temp file. When you start your database, Oracle will automatically try to add any missing temp files to the database. Oracle won't be able to do this if the directory structure on the destination server is different from that of the source server. In this scenario, you will have to add any missing temp files manually. To do this, first take offline the temporary tablespace temp file. The file definition from the originating database is taken offline like so:

```
SQL> alter database tempfile '/u01/dbfile/TRG/temp01.dbf' offline;
SQL> alter database tempfile '/u01/dbfile/TRG/temp01.dbf' drop;
```

Next, add a temporary tablespace file to the *TEMP* tablespace that matches the directory structure of the destination database server:

```
SQL> alter tablespace temp add tempfile '/u01/dbfile/DUP/temp01.dbf'
     size 100m;
```

You can run the *REPORT SCHEMA* command to verify that all files are in the correct locations.

16. Rename the database (optional). If you need to rename the database to reflect the name for a development or test database, create a trace file that contains the *CREATE CONTROLFILE* statement and use it to rename your database. For details on how to rename a database, see the next section "Renaming a Database."

■ **Tip** If you don't rename the database, be careful about connect and resync operations to the same recovery catalog used by the original/source database. This causes confusion in the recovery catalog as to which is the real source database, which may jeopardize your ability to recover and restore the real source database.

Also keep in mind that other steps may be required for your environment depending on your standards. For example, you might want to ensure the database SID is listed in the *oratab* file, or you might require the use of an SPFILE, enabling a password file, changing passwords, taking a backup, adding entries into Oracle Net files, and so on.

Renaming a Database

This section shows you how to rename a database. If you're working with a critical database, make sure you have a good backup of the data files, control files, and any relevant archive redo logs before you change the name.

Two different ways of renaming your database are described next. The first renaming method walks you through the manual steps. The second technique describes renaming a database with the Oracle NID utility. If you need to assign a new DBID to the renamed database, then you should use the NID utility.

Manual

In this example, the database is renamed from *TRG* to *DUP*. The steps for manually renaming your database are as follows:

1. Generate a trace file that contains the SQL command to recreate the control files:

```
SQL> alter database backup controlfile to trace as '/tmp/cf.sql' resetlogs;
```

2. Shut down the database:

```
SQL> shutdown immediate;
```

3. Modify the */tmp/cf.sql* trace file; be sure to specify *SET DATABASE "<NEW DATABASE NAME>"* in the top line of the output:

```
CREATE CONTROLFILE REUSE SET DATABASE "DUP" RESETLOGS ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 4
    MAXDATAFILES 1024
    MAXINSTANCES 1
    MAXLOGHISTORY 876
```

```
LOGFILE
  GROUP 1 '/u01/oraredo/DUP/redo01a.rdo' SIZE 50M BLOCKSIZE 512,
  GROUP 2 '/u01/oraredo/DUP/redo02a.rdo' SIZE 50M BLOCKSIZE 512
DATAFILE
  '/u01/dbfile/DUP/system01.dbf',
  '/u01/dbfile/DUP/sysaux01.dbf',
  '/u01/dbfile/DUP/undotbs01.dbf',
  '/u01/dbfile/DUP/users01.dbf',
  '/u01/dbfile/DUP/repdata.dbf',
  '/u01/dbfile/DUP/repidx.dbf'
CHARACTER SET AL32UTF8;
```

If you don't specify *SET DATABASE* in the top line of this script, when you run the script (as shown later in this example) you'll receive an error such as this:

```
ORA-01161: database name ... in file header does not match...
```

4. Create an *init.ora* file that matches the new database name:

```
$ cd $ORACLE_HOME/dbs
$ cp init<old_sid>.ora init<new_sid>.ora
```

In this example, the prior line of code looks like this:

```
$ cp initTRG.ora initDUP.ora
```

5. Modify the *DB_NAME* variable within the new *initDUP.ora* file (in this example, it's set to *DUP*):

```
db_name='DUP'
```

6. If the instance with the old SID is still running, then shut it down (*TRG* in this example):

```
SQL> shutdown immediate;
```

7. Set the *ORACLE_SID* OS variable to reflect the new SID name (in this example, it's set to *DUP*):

```
$ export ORACLE_SID=DUP
$ echo $ORACLE_SID
DUP
```

8. Start up the instance in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
```

9. Run the trace file to recreate the control file:

```
SQL> @/tmp/cf.sql
```

If successful you should see:

```
Control file created.
```

■ **Note** In this example, the control files already exist in the location specified by the *CONTROL_FILES* initialization parameter; therefore, the *REUSE* parameter is used in the *CREATE CONTROL FILE* statement.

---

10. Open the database with *OPEN RESETLOGS*:

```
SQL> alter database open resetlogs;
```

If successful you should see:

```
Database altered.
```

11. As a last step, ensure that your temporary tablespace exists:

```
SQL> ALTER TABLESPACE TEMP ADD TEMPFILE '/u01/dbfile/DUP/temp01.dbf'
     SIZE 104857600  REUSE AUTOEXTEND OFF;
```

You now have a database that is a copy of the original database. All the data files, control files, archive redo logs, and online redo logs are in the new locations, and the database has a new name. Now would be a good time to take a backup of the newly renamed database, recreate the password file (if using), and modify service name values in Oracle Net files.

If you need to assign the database a new DBID, then you can use a utility such as NID to accomplish this.

NID

This section describes using the NID utility to rename a database. This procedure will rename the database and assign it a new DBID.

1. First start the database in mount mode:

```
$ sqlplus / as sysdba
SQL> startup mount;
```

2. Now, from the operating system command line run the NID utility. This renames the database from *DUP* to *DUP_NEW*. You have to modify this as appropriate for the database names and passwords in your environment:

```
$ nid target=sys/foo dbname=DUP_NEW
```

In the output you should see a line similar to this:

```
Change database ID and database name DUP to DUP_NEW? (Y/[N]) =>
```

Respond with *Y* if you wish to proceed. Here's a sample of the output for this example:

```
Proceeding with operation
Changing database ID from 1251088236 to 1191846239
Changing database name from DUP to DUP_NEW
    Control File /u01/dbfile/DUP/control01.ctl - modified
    Control File /u01/dbfile/DUP/control02.ctl - modified
    Datafile /u01/dbfile/DUP/system01.db - dbid changed, wrote new name
...
All previous backups and archived redo logs for this database are unusable.
Database has been shut down, open database with RESETLOGS option.
Successfully changed database name and ID.
DBNEWID - Completed successfully.
```

Next create an initialization file that corresponds to the new database name:

```
$ cd $ORACLE_HOME/dbs
$ cp initDUP.ora initDUP_NEW.ora
```

3. Modify the *DB_NAME* parameter in the *initDUP_NEW.ora* to reflect the new database name:

```
db_name=DUP_NEW
```

4. Set *ORACLE_SID* to reflect the new database name:

```
$ export ORACLE_SID=DUP_NEW
```

5. Mount the database:

```
$ sqlplus / as sysdba
SQL> startup mount;
```

6. Open the database with the *OPEN RESETLOGS* clause:

```
SQL> alter database open resetlogs;
```

You can verify that the database has the new DBID assigned to it using the following:

```
SQL> select dbid from v$database;

      DBID
----------
1191846239
```

Now would be a good time to take a backup of the newly renamed database, recreate the password file (if using), and modify service name values in Oracle Net files.

■ **Tip** See MOS note 863800.1 for more details regarding NID. You can leverage NID to only change the DBID (and not the database name), or you can use NID to change only change the database name (and not the DBID).

Replicating with Data Pump Across a Network Link

Data Pump is a powerful and flexible tool for moving data from one environment to another. This utility has significant advantages over other data replication methods, especially in the following situations:

- You need the ability to replicate at the table, tablespace, schema, or database level of granularity. For example, with Data Pump it's simple to copy a schema and all its objects, with or without the data, from one database to another. Furthermore, you can also filter and transform the data during the replication.
- You need cross-platform replication. Any combination of cross-platform replication is possible. For instance, you can move data seamlessly between any two operating systems (Linux, Solaris, Windows, and so on) where Oracle is installed.
- You need to export data and import it into a database with the same version, or to any higher version. Some shops use this as a database upgrade mechanism.

A real-world example will help illustrate the utility. Suppose you have two database environments—a production database running on a Solaris box and a test database running on a Linux server. Your manager comes to you with these instructions:

- Make a copy of the production database on the Solaris box and import the copy into the test database on the Linux server.
- Change the names of the schemas when importing to reflect naming standards in the test environment.

First consider the steps required to transfer data from one database to another when using the old exp/imp utilities. The steps would look something like this:

1. Export the production database (which creates a dump file on the database server).
2. Copy the dump file to the testing database server.
3. Import the dump file into the testing database.

You can perform those same steps using Data Pump. However, Data Pump provides a much more efficient and transparent method for executing those steps. If you have direct network connectivity between the source and destination database servers, you can import directly into the destination database over the network without having to create and/or copy any dump files. Furthermore, you can rename schemas on the fly as you perform the import. Additionally, it doesn't matter if the source database is running on a different operating system than that of the destination database. Figure 2-3 illustrates the environment and required steps.

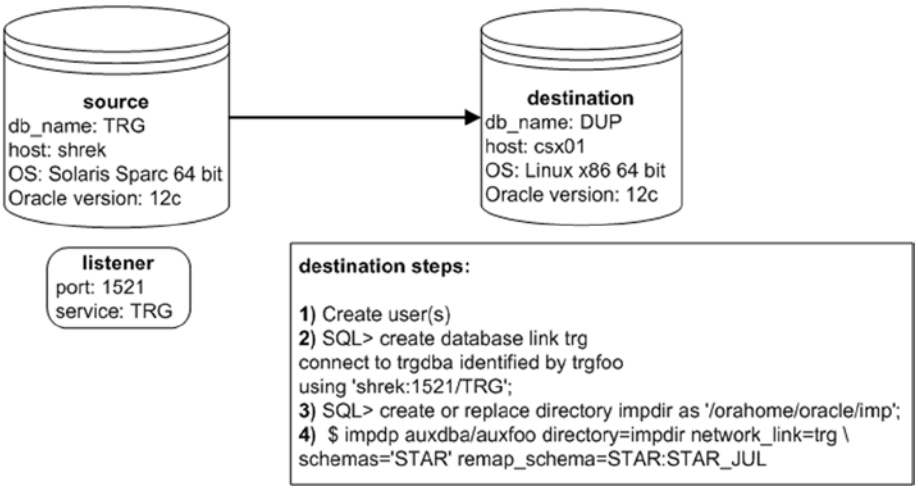

Figure 2-3. *Data Pump export and import across a network link*

For this example, in the source database there's a schema named *STAR*. You want to move this user into the destination database and rename it to *STAR_JUL*. Also assume that the tablespace names are the same in both the source and the destination databases.

Notice that all of the following steps are performed on the destination server. No steps are required on the source server.

1. In the destination database, create a user to be imported into. Here is a sample script that creates the user:

```
define star_user=star_jul
define star_user_pwd=star_jul_pwd
--
create user &&star_user identified by &&star_user_pwd;
grant create session, create table, create procedure to &&star_user;
```

2. In the destination database, create a database link that points to the source database. The remote user (in the source database) referenced in the *CREATE DATABASE LINK* statement must have a privileged role granted to it in the source database (this minimally needs to be the *DATAPUMP_EXP_FULL_DATABASE* role). Log into SQL as the user that you'll use later to import the data. You want to ensure that the database link is created with the same schema used when importing. In this example, *AUXDBA* is the user in the destination database:

```
$ sqlplus auxdba/auxfoo
```

Here is a sample *CREATE DATABASE LINK* script:

```
SQL> create database link trg
connect to trgdba identified by trgfoo
using 'shrek:1521/TRG';
```

3. In the destination database, create a directory object that points to the location where you want your log file to go. You'll have to modify this to match a directory in your environment:

```
SQL> create or replace directory dpdir as '/orahome/oracle/dpdir';
```

With the prior directory object, if you're not using a privileged user (e.g., a user that has been granted the DBA role) you may need to additionally grant *READ* and *WRITE* privileges on the directory to the user. For instance:

```
SQL> grant read, write on directory dpdir to auxdba;
```

4. Run the *import* command on the destination box. This command references the remote database via the *NETWORK_LINK* parameter. This command also instructs Data Pump to map the target database schema to the newly created user in the destination database:

```
$ impdp auxdba/auxfoo directory=dpdir network_link=trg \
schemas='STAR' remap_schema=STAR:STAR_JUL
```

This technique allows you to move large amounts of data between disparate databases without having to create or copy any dump files or data files. You can also rename schemas on the fly via the *REMAP_SCHEMA* parameter. If the tablespace names weren't the same on both the source and destination, you can use the *REMAP_TABLESPACE* parameter to have tables placed in different tablespaces in the destination database. This is a very powerful Data Pump feature that lets you efficiently transfer data between disparate databases.

■ **Tip** For a complete description of Data Pump's features, see *Pro Oracle Database 12c Administration* available from Apress.

Replicating with Data Pump Transportable Tablespaces

Oracle provides a mechanism for copying data files from one database to another in conjunction with using Data Pump to transport the associated metadata. This is known as the *transportable tablespace* feature. The amount of time this task requires is directly proportional to the time it takes to copy the data files to the destination server. In this scenario both the source and destination servers have the same operating system platform. Figure 2-4 shows the systems and the steps required to transport tablespaces for this scenario.

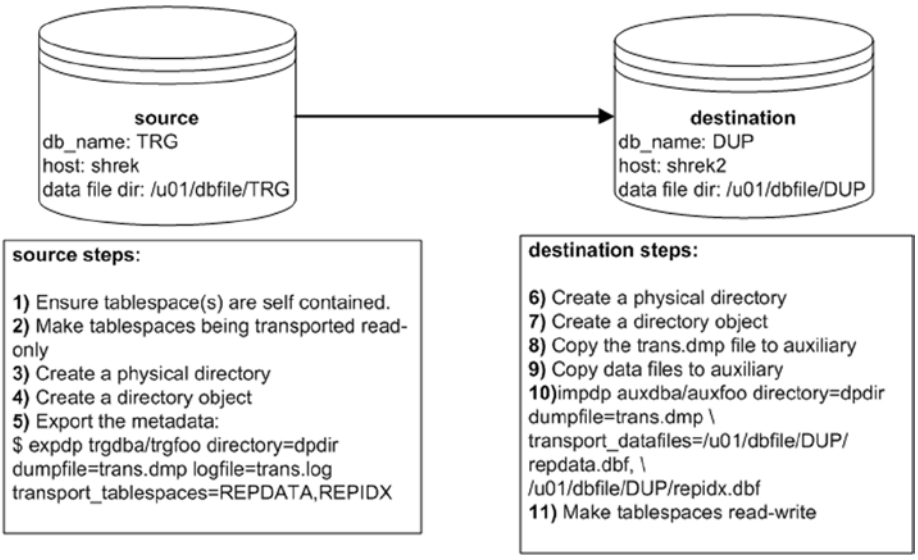

Figure 2-4. *Using Data Pump with transportable tablespaces*

The steps depicted in Figure 2-4 are described in detail next.

1. First ensure that the tablespaces being transported are self-contained. These are some common violations of the self-contained rule:
   - An index in one tablespace can't point to a table in another tablespace that isn't in the set of tablespaces being transported.
   - A foreign-key constraint is defined on a table in a tablespace that references a primary-key constraint on a table in a tablespace that isn't in the set of tablespaces being transported.

Run the following check on the source database to see if the set of tablespaces being transported violates any of the self-contained rules:

```
SQL> exec dbms_tts.transport_set_check('REPDATA,REPIDX', TRUE);
```

Now, see if Oracle detected any violations:

```
SQL> select * from transport_set_violations;
```

If you don't have any violations, you should see this:

```
no rows selected
```

If you do have violations, such as an index that is built on a table that exists in a tablespace not being transported, then you'll have to rebuild the index in a tablespace that *is* being transported. Be aware that detecting and resolving violations can lead to other tablespaces being required to be added in the transportable set. This can turn into a much bigger task than one might initially anticipate.

2. Make the tablespaces being transported read-only. In this example the tablespaces *REPDATA* and *REPIDX* are going to be transported:

```
SQL> alter tablespace repdata read only;
SQL> alter tablespace repidx read only;
```

3. Create a physical directory on disk:

```
$ mkdir /orahome/oracle/dpdir
```

4. Create a directory object that points to a directory that exists on disk:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

5. Use Data Pump export on the target server to extract the metadata for the tablespaces being transported (in this example, *REPDATA* and *REPIDX*). When exporting, use a user that has been assigned the DBA role:

```
$ expdp trgdba/trgfoo directory=dpdir \
dumpfile=trans.dmp logfile=trans.log \
transport_tablespaces=REPDATA,REPIDX
```

6. On the destination server, create a directory to hold the Data Pump export file (created in a previous step):

```
$ mkdir /orahome/oracle/dpdir
```

7. On the destination server, create a directory object that points to the physical exist directory.

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

8. Copy the Data Pump export dump file from the source server to the destination server. The following line of code uses the Linux/UNIX *scp* command to copy the dump file. This is initiated from the destination server:

```
$ scp oracle@shrek:/orahome/oracle/dpdir/trans.dmp /orahome/oracle/dpdir
```

9. Copy the data file(s) from the source server to the destination server. Place the files in the directory where you want them on the destination database server. The following line of code uses the Linux/UNIX *scp* command to copy the data files. This is initiated from the destination server:

```
$ scp oracle@shrek:/u01/dbfile/TRG/rep*.dbf /u01/dbfile/DUP
```

10. On the destination server, import the metadata into the destination database. Use the following parameter file to import the metadata for the data files being transported. When importing, use a user that has been assigned the DBA role:

```
$ impdp auxdba/auxfoo directory=dpdir dumpfile=trans.dmp \
transport_datafiles=/u01/dbfile/DUP/repdata.dbf, \
/u01/dbfile/DUP/repidx.dbf
```

Additionally, ensure that the owner(s) of any tables or indexes within the tablespaces being transported exist(s) in the destination database. If the owning schema doesn't exist in the destination database, you'll receive this error:

```
ORA-29342: user ... does not exist in the database
```

If everything goes well, you should see some output indicating success:

```
job "AUXDBA"."SYS_IMPORT_TRANSPORTABLE_01" successfully completed ...
```

11. As a final step, you may want to change the tablespaces back to read-write. Depending on your requirements, you may want to perform this on both the destination and the source databases:

```
SQL> alter tablespace repdata read write;
SQL> alter tablespace repidx read write;
```

If the data files that are being transported have a block size different from that of the destination database, then you must modify your initialization file (or use an *ALTER SYSTEM* command) and add a buffer pool that contains the block size of the source database. For example, to add a 16KB buffer cache, place this in the initialization file:

```
db_16k_cache_size=200M
```

You can check a tablespace's block size via this query:

```
SQL> select tablespace_name, block_size from dba_tablespaces;
```

The transportable tablespace mechanism allows you to quickly move data files between databases. It's an appropriate method for moving data in data warehouse–type environments where you might have a staging database that periodically needs data transferred to a reporting database.

■ **Note** To export transportable tablespaces, you must use Oracle Enterprise Edition. You can use other editions of Oracle to import transportable tablespaces.

RMAN Replication Using Transportable Tablespaces

You can use RMAN in conjunction with transportable tablespaces to copy data files from one database to a different database. In RMAN terminology the source database is referred to as the *target* and the destination database is referred to as the *auxiliary*. The target server can have the same operating system as the auxiliary server, or the target server can have a different operating system than the auxiliary server.

When the operating system is the same, you can issue the RMAN *TRANSPORT TABLESPACE* command to generate the tablespace metadata dump file. In this mode, the advantage to using RMAN is that you can keep the live data files online during the procedure (meaning you don't have to place the tablespaces in read-only mode like Data Pump requires when transporting tablespaces).

When the operating systems are different (such as different endian formats), you'll have to use the RMAN *CONVERT* command to create data files that can be used by a database running on a different operating system with a different endian format. In this mode, the tablespaces must be in read-only mode while the RMAN command is running.

Scenarios for transporting between two servers that have the same operating system and between two that have different operating systems are covered in the following sections.

Same Operating System

Using RMAN to transport tablespaces between servers that have the same operating system platform is fairly straightforward. As mentioned previously, in this configuration you can transport data files while the tablespaces are online. RMAN achieves this high availability by utilizing RMAN backups and archive logs while creating the transportable tablespace dump files. This means you must have a valid RMAN backup of the target database. If you don't have a backup, when you attempt to run the *TRANSPORT TABLESPACE* command you'll receive this error:

```
RMAN-06024: no backup or copy of the control file found to restore
```

The target database must be in *archivelog* mode also, if you attempt to run *TRANSPORT TABLESPACE* on a *noarchivelog* mode database you'll receive the following error:

```
RMAN-05004: target database log mode is NOARCHIVELOG
```

Figure 2-5 illustrates the basic steps involved when using RMAN to transport tablespaces from one server and database to another when the servers are of the same operating system.

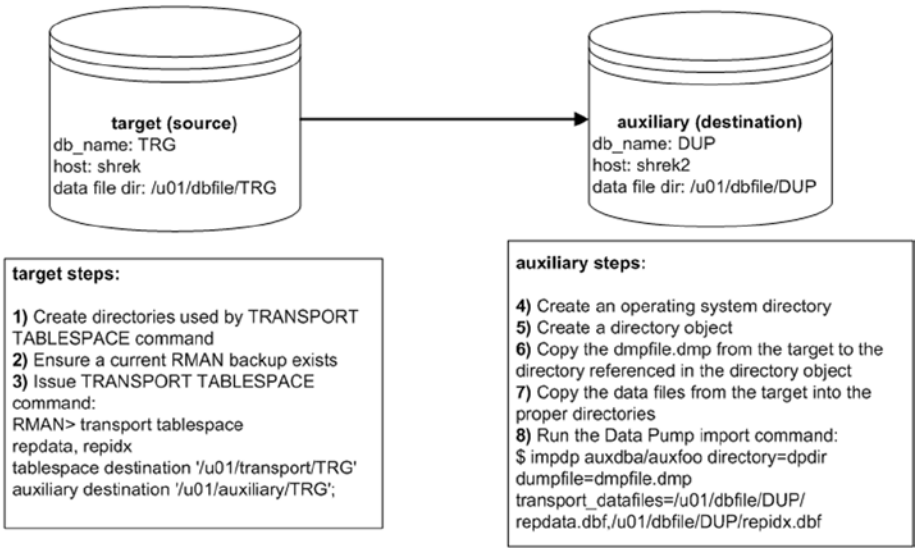

Figure 2-5. *RMAN and transportable tablespaces (same platforms)*

The steps shown in Figure 2-5 are described in detail next.

1. On the target server, create two directories, one for the transport files that RMAN will generate and one for the auxiliary files that RMAN will generate (in a future step). You'll have to modify this as appropriate for your environment:

```
$ mkdir -p /u01/transport/TRG
$ mkdir -p /u01/auxiliary/TRG
```

2. Ensure you have a current backup of the target database:

```
$ rman target /
RMAN> list backup;
```

3. While connected via RMAN to the target database, issue the *TRANSPORT TABLESPACE* command for the tablespaces you wish to transport (*REPDATA* and *REPIDX* in this example):

```
$ rman target /
RMAN> transport tablespace repdata, repidx
tablespace destination '/u01/transport/TRG'
auxiliary destination '/u01/auxiliary/TRG';
```

In the prior command, the *tablespace destination* is where RMAN will place files that can be used to transport the tablespaces. The *auxiliary destination* is used by RMAN to create files associated with a temporary auxiliary database that is used to generate the transportable tablespace files. This temporary auxiliary database will be dropped after the transportable tablespace files have been generated and placed in the transport directory. Don't confuse this temporary auxiliary database with the auxiliary database that you'll be transporting the tablespaces to.

4. On the auxiliary server, from the operating system create an operating system directory:

```
$ mkdir /orahome/oracle/dpdir
```

This is the directory in which you're going to place the *dmpfile.dmp* (from the target). If you've worked through examples in previous sections in this book, this directory may already exist, and that's fine.

5. On the auxiliary server, create a directory object that points to the physical operating system directory created in the prior set:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

If you've worked through previous examples in this book, then this directory object may already exist, and that's fine. Just make sure the directory object *DPDIR* points to the */orahome/oracle/dpdir* directory.

6. Copy the *dmpfile.dmp* from the target server to the auxiliary server. This example uses the Linux/UNIX *scp* command initiated from the auxiliary server:

```
$ scp oracle@shrek:/u01/transport/TRG/dmpfile.dmp /orahome/oracle/dpdir
```

7. Copy the database files from the target server to the auxiliary server. This example uses the Linux/UNIX *scp* command initiated from the auxiliary server:

```
$ scp oracle@shrek:/u01/transport/TRG/repdata.dbf  /u01/dbfile/DUP
$ scp oracle@shrek:/u01/transport/TRG/repidx.dbf  /u01/dbfile/DUP
```

8. On the auxiliary server, run the Data Pump import command to import the metadata for the transported tablespaces:

```
$ impdp auxdba/auxfoo directory=dpdir dumpfile=dmpfile.dmp \
transport_datafiles=/u01/dbfile/DUP/repdata.dbf,/u01/dbfile/DUP/repidx.dbf
```

When finished, you should have the tablespaces transported into the auxiliary (destination) database.

Cross-Platform Replication

In some scenarios you can use RMAN commands such as *DUPLICATE*, *RESTORE*, and *RECOVER* when the target server uses a different operating system than the auxiliary server. RMAN supports these operations only for the following operating system combinations that have the same endian format:

- To and from Solaris x86-64 and Linux x86-64
- To and from HP-PA and HP-IA
- To and from Windows 64-bit (IA or Itanium) and Windows 64-bit (AMD or x86-64)
- To and from Linux and Windows

Additionally, these types of operations are only supported when the Oracle version in use is the same on both the target and auxiliary databases. Also, the two environments must be at the same patch level.

■ **Tip** See MOS note 1079563.1 for details on operations allowed when different platforms are in use for the target and auxiliary servers.

If an operating system combination is not listed in the prior bulleted list then you must use other supported migration procedures, such as transportable tablespace, transportable database, or Data Pump export/import. See the sections in this chapter "Different Operating System (Convert Tablespace)" and "Different Operating System (Convert Data File)" for examples of using RMAN to transport a tablespace (and associated data files) between operating systems with different endian formats.

Different Operating System (Convert Tablespace)

When transporting between different operating systems that have different endian formats, the *CONVERT TABLESPACE* command facilitates the converting of the data files from the source operating system platform to the destination platform. After the data files have been converted, they can be copied to a host of different operating systems with the different endian format.

I recently worked on a project moving a database from a Solaris SPARC 64-bit server to a Linux x-86 64-bit host. The steps involved in using *CONVERT TABLESPACE* to convert the data files are depicted in Figure 2-6.

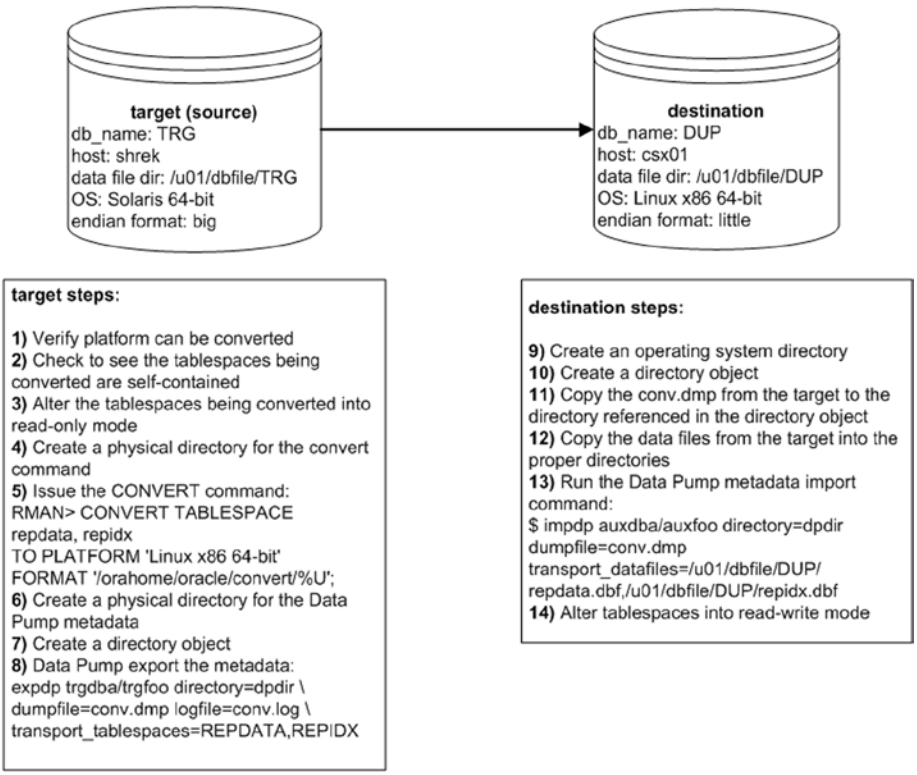

Figure 2-6. *RMAN and transportable tablespaces (different platforms)*

Details of the steps shown in Figure 2-6 follow next.

1. On the target database, confirm the platforms that tablespaces can be converted to via the following SQL:

```
SQL> SELECT platform_id, platform_name, endian_format
FROM V$TRANSPORTABLE_PLATFORM
WHERE UPPER(platform_name) LIKE 'LINUX%';
```

Here is some sample output:

```
PLATFORM_ID PLATFORM_NAME                  ENDIAN_FORMAT
----------- ------------------------------ --------------------
         10 Linux IA (32-bit)              Little
         11 Linux IA (64-bit)              Little
         13 Linux x86 64-bit               Little
```

In this case, the target server can convert data files to be used on the destination Linux x86 64-bit box with the little endian format.

2. On the target database, run the following check to see if the set of tablespaces being transported violates any of the self-contained rules:

```
SQL> exec dbms_tts.transport_set_check('REPDATA,REPIDX', TRUE);
```

Now, see if Oracle detected any violations:

```
SQL> select * from transport_set_violations;
```

If you don't have any violations, you should see this:

```
no rows selected
```

If you do have violations, such as an index that is built on a table that exists in a tablespace not being transported, then you'll have to rebuild the index in a tablespace that *is* being transported. If you need more details on the *DBMS_TTS.TRANSPORT_SET_CHECK* procedure, refer to the *Oracle Database PL/SQL Package and Types Reference* guide available on Oracle's technology network website.

3. Place the target tablespaces in read-only mode:

```
SQL> alter tablespace repdata read only;
SQL> alter tablespace repidx read only;
```

4. Create a directory on the target server to hold the data files that will be converted. You'll have to modify this as appropriate for your environment:

```
$ mkdir /orahome/oracle/convert
```

5. Connect to the target database and run the *CONVERT TABLESPACE* command:

```
$ rman target /
RMAN> CONVERT TABLESPACE repdata, repidx
TO PLATFORM 'Linux x86 64-bit'
FORMAT '/orahome/oracle/convert/%U';
```

You should now have the converted data files in the specified directory:

```
$ ls /orahome/oracle/convert
```

Here is some sample output:

```
data_D-TRG_I-1251088236_TS-REPDATA_FNO-5_brprfa57
data_D-TRG_I-1251088236_TS-REPIDX_FNO-6_bsprfa57
```

6. On the target server, create a physical directory on disk:

```
$ mkdir /orahome/oracle/dpdir
```

7. On the target database create a directory object that points to a directory that exists on disk:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

8. Use Data Pump *export* on the target server to export the metadata for the tablespaces being transported:

```
$ expdp trgdba/trgfoo directory=dpdir \
dumpfile=conv.dmp logfile=conv.log \
transport_tablespaces=REPDATA,REPIDX
```

9. Now, on the destination server create a directory to contain files that will be copied to it:

```
$ mkdir /orahome/oracle/dpdir
```

10. Create a directory object to point to the physical directory created in the prior step:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

11. Copy the Data Pump export dump file from the target server to the destination server. The following line of code uses the Linux/UNIX *scp* command to copy the dump file. This is initiated from the destination server:

```
$ scp oracle@shrek:/orahome/oracle/dpdir/conv.dmp /orahome/oracle/dpdir
```

12. Copy the data file(s) from the target server to the destination server. Place the files in the directory where you want them in the destination database server. The following line of code uses the Linux/UNIX *scp* command to copy the data files. This is initiated from the destination server:

```
$ scp oracle@shrek:/orahome/oracle/convert/data_D-TRG_I-1251088236_TS-
REPDATA_FNO-5_brprfa57 /u01/dbfile/DUP/repdata.dbf

$ scp oracle@shrek:/orahome/oracle/convert/data_D-TRG_I-1251088236_TS-
REPIDX_FNO-6_bsprfa57 /u01/dbfile/DUP/repidx.dbf
```

13. On the destination server, import the metadata into the destination database:

```
$ impdp auxdba/auxfoo directory=dpdir dumpfile=conv.dmp \
transport_datafiles=/u01/dbfile/DUP/repdata.dbf,/u01/dbfile/DUP/repidx.dbf
```

You should now have the converted tablespaces (and associated data files) in the destination database.

14. As a final step, you may want to change the tablespaces back to read-write:

```
SQL> alter tablespace repdata read write;
SQL> alter tablespace repidx read write;
```

Different Operating System (Convert DataFile)

You can also transport data across platforms with different endian formats using the RMAN *CONVERT DATAFILE* command. This example performs a conversion between Solaris 64-bit with big endian format to a Linux server with little endian format, with the conversion taking place on the destination server. The environment used and steps required are shown in Figure 2-7.

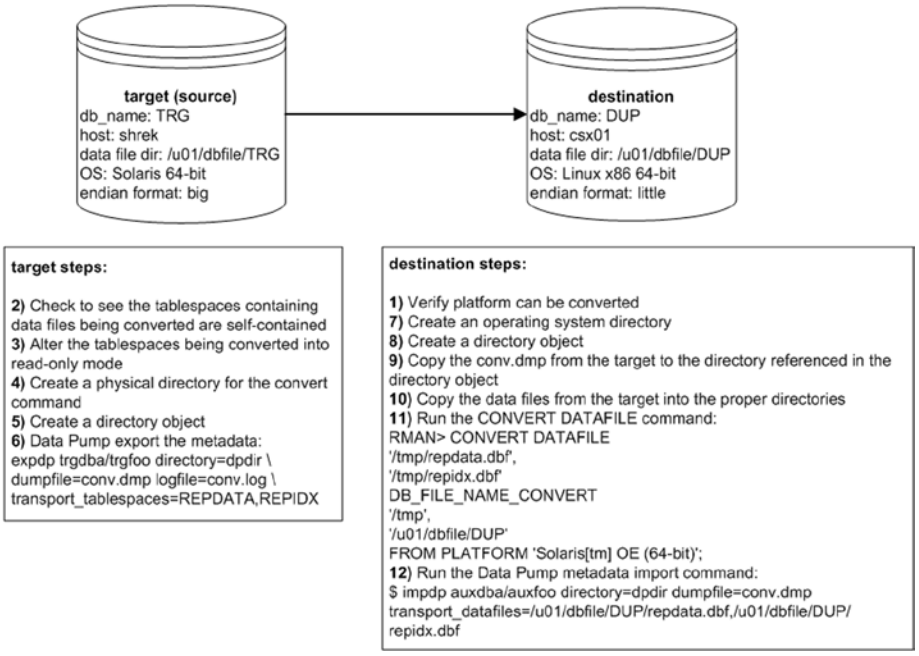

Figure 2-7. *Converting data files between operating systems with different endian formats*

Details of the steps shown in Figure 2-7 follow next.

1. On the destination database (on the Linux server, in this example), confirm the platform can be converted from via the following SQL:

```
SQL> SELECT platform_id, platform_name, endian_format
FROM V$TRANSPORTABLE_PLATFORM
WHERE UPPER(platform_name) LIKE 'SOLARIS%';
```

Here is some sample output:

```
PLATFORM_ID PLATFORM_NAME                          ENDIAN_FORMAT
----------- ------------------------------------- --------------
          1 Solaris[tm] OE (32-bit)               Big
          2 Solaris[tm] OE (64-bit)               Big
         17 Solaris Operating System (x86)        Little
         20 Solaris Operating System (x86-64)     Little
```

In this case, the destination (Linux) server is capable of converting data files from a Solaris box with the big endian format.

2. On the source database (Solaris, in this example), run the following check to see if the set of tablespaces being transported violates any of the self-contained rules:

```
SQL> exec dbms_tts.transport_set_check('REPDATA,REPIDX', TRUE);
```

Now, see if Oracle detected any violations:

```
SQL> select * from transport_set_violations;
```

If you don't have any violations, you should see this:

```
no rows selected
```

If you do have violations, such as an index that is built on a table that exists in a tablespace not being transported, then you'll have to rebuild the index in a tablespace that *is* being transported.

3. Place the source tablespaces in read-only mode:

```
SQL> alter tablespace repdata read only;
SQL> alter tablespace repidx read only;
```

4. On the source server, create a physical directory on disk:

```
$ mkdir /orahome/oracle/dpdir
```

5. On the source database, create a directory object that points to a directory that exists on disk:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

6. Use Data Pump on the target server to export the metadata for the tablespaces being transported:

```
$ expdp trgdba/trgfoo directory=dpdir \
dumpfile=conv.dmp logfile=conv.log \
transport_tablespaces=REPDATA,REPIDX
```

7. On the destination Linux server, create a directory to place the dump file in:

```
$ mkdir /orahome/oracle/dpdir
```

8. On the destination Linux server, create a directory object that points to the directory holding the Data Pump metadata:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

9. Copy the *conv.dmp* from the target server to the destination server. This example uses the Linux/UNIX *scp* command initiated from the destination server:

```
$ scp oracle@shrek:/orahome/oracle/dpdir/conv.dmp /orahome/oracle/dpdir
```

10. Copy the database files from the source server to the destination server. This example uses the Linux/UNIX *scp* command initiated from the destination server:

```
$ scp oracle@shrek:/u01/dbfile/TRG/repdata.dbf /tmp
$ scp oracle@shrek:/u01/dbfile/TRG/repidx.dbf  /tmp
```

11. On the destination Linux server, connect to RMAN and run the *CONVERT DATAFILE* command:

```
$ rman target /
RMAN> CONVERT DATAFILE
'/tmp/repdata.dbf',
'/tmp/repidx.dbf'
DB_FILE_NAME_CONVERT
'/tmp',
'/u01/dbfile/DUP'
FROM PLATFORM 'Solaris[tm] OE (64-bit)';
```

12. Next, run Data Pump import so as to import the metadata associated with the data files being transported:

```
$ impdp auxdba/auxfoo directory=dpdir dumpfile=conv.dmp \
transport_datafiles=/u01/dbfile/DUP/repdata.dbf, \
/u01/dbfile/DUP/repidx.dbf
```

This imports into the destination database the metadata for the tablespaces (and associated data files) being converted. You can verify the data files exist in the data dictionary via:

```
SQL> select name from v$datafile where name like '%rep%';

NAME
--------------------------------------------------
/u01/dbfile/DUP/repdata.dbf
/u01/dbfile/DUP/repidx.dbf
```

You may want to place the newly converted tablespaces into read-write mode at this point:

```
SQL> alter tablespace repdata read write;
SQL> alter tablespace repidx read write;
```

Moving Data with External Tables

External tables are primary used to load data from *csv* files into the database. External tables can also be used to select data from a regular database table and create a binary dump file. The dump file is platform independent and can be used to move large amounts of data between servers of different platforms and different endian formats.

You can also encrypt or compress data, or both, when creating the dump file. Doing so provides you with an efficient and secure way of transporting data between database servers.

Figure 2-8 illustrates the components involved in using an external table to unload and load data. On the target (source) database, create a dump file using an external table that selects data from a table named *INV*. After it's created, copy the dump file to the auxiliary (destination) server and subsequently load the file into the database using an external table.

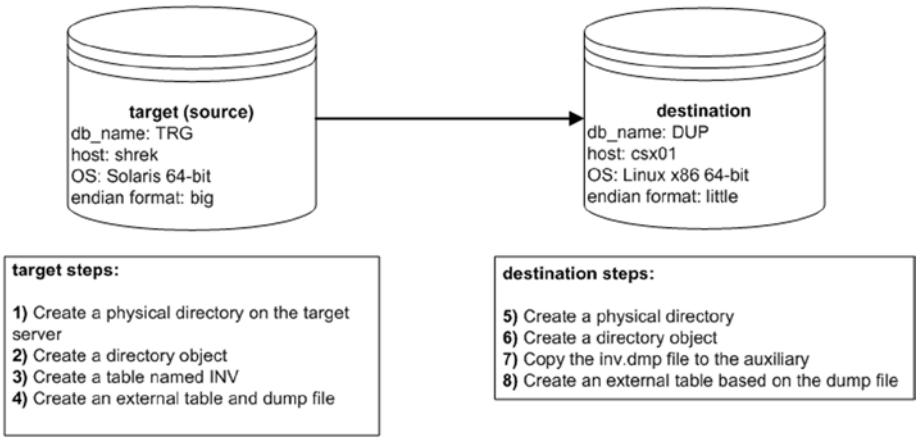

Figure 2-8. *Using external tables to unload and load data*

A small example illustrates the technique of using an external table to unload data. Here are the steps required:

1. Create a physical directory on the target server:

```
$ mkdir /orahome/oracle/dpdir
```

2. Create a directory object that points to the physical directory:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

3. Create a table named *INV* and insert some data:

```
SQL> CREATE TABLE inv
(inv_id NUMBER,
 inv_desc VARCHAR2(30));
SQL> insert into inv values (1, 'test data');
SQL> commit;
```

4. Use the *CREATE TABLE...ORGANIZATION EXTERNAL...AS SELECT* statement to unload data from the database into the dump file. Use the *ORACLE_DATAPUMP* access driver of the *CREATE TABLE...ORGANIZATION EXTERNAL* statement. This example unloads the *INV* table's contents into the *inv.dmp* file:

```
CREATE TABLE inv_et
ORGANIZATION EXTERNAL (
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY dpdir
  LOCATION ('inv.dmp')
)
AS SELECT * FROM inv;
```

The previous command creates two things:

- An external table named *INV_ET* based on the structure and data within the *INV* table
- A platform-independent dump file named *inv.dmp*

5. Now, on the destination server create a physical directory:

```
$ mkdir /orahome/oracle/dpdir
```

6. On the destination database, create a directory object that references the physical directory:

```
SQL> create directory dpdir as '/orahome/oracle/dpdir';
```

7. Now, you can copy the *inv.dmp* file to a separate database server and base an external table on this dump file. The following example uses the Linux/UNIX *scp* command to copy the file. This command is initiated from the destination server:

```
$ scp oracle@shrek:/orahome/oracle/dpdir/inv.dmp  /orahome/oracle/dpdir
```

The remote server (to which you copy the dump file) can be a platform different from that of the server on which you created the file. For example, you can create a dump file on a Windows box, copy to a Linux/UNIX server, and select from the dump file via an external table.

8. On the destination database, create an external table that points at the dump file. In this example the external table is named *INV_DW*:

```
SQL> CREATE TABLE inv_dw
(inv_id number
,inv_desc varchar2(30))
ORGANIZATION EXTERNAL (
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY dpdir
  LOCATION ('inv.dmp'));
```

After it's created, you can access the external table data from SQL*Plus:

```
SQL> select * from inv_dw;
    INV_ID INV_DESC
---------- ------------------------------
         1 test data
```

You can also create and load data into regular tables using the dump file:

```
SQL> create table inv as select * from inv_dw;
```

This provides a simple and efficient mechanism for transporting data from one platform to another.

■ **Tip** For complete details on external tables, see *Expert Oracle Database Architecture*, available from Apress.

Enabling Parallelism

To maximize the unload performance when you create a dump file via an external table, use the *PARALLEL* clause. This example creates two dump files in parallel:

```
CREATE TABLE inv_et
ORGANIZATION EXTERNAL (
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY dpdir
  LOCATION ('inv1.dmp','inv2.dmp')
)
PARALLEL 2
AS SELECT * FROM inv;
```

To access the data in the dump files, create a different external table that references the two dump files:

```
CREATE TABLE inv_dw
(inv_id number
,inv_desc varchar2(30))
ORGANIZATION EXTERNAL (
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY dpdir
  LOCATION ('inv1.dmp','inv2.dmp'));
```

You can now use this external table to select data from the dump files:

```
SQL> select * from inv_dw;
```

Enabling Compression

You can create a compressed dump file via an external table. For example, use the *COMPRESS* option of the *ACCESS PARAMETERS* clause:

```
CREATE TABLE inv_et
ORGANIZATION EXTERNAL (
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY dpdir
  ACCESS PARAMETERS (COMPRESSION ENABLED BASIC)
  LOCATION ('inv.dmp')
)
AS SELECT * FROM inv;
```

In Oracle 12c there are four levels of compression: *BASIC*, *LOW*, *MEDIUM*, and *HIGH*. Before using compression, ensure that the *COMPATIBLE* initialization parameter is set to 12.0.0 or higher. The *LOW*, *MEDIUM*, and *HIGH* levels of compression require Oracle Enterprise Edition, along with the Advanced Compression option.

■ **Tip** You can also enable encryption when transporting data via external tables. See the *Oracle Advanced Security Administrator's Guide*, which can be freely downloaded from the Technology Network area of the Oracle web site (http://otn.oracle.com), for full details on implementing encryption.

Summary

This chapter lays the foundation for understanding how you can move data from one database environment to another. This chapter discussed several different manual techniques for moving data:

- Cloning from a cold backup
- Restoring from an RMAN backup
- Replicating with Data Pump across a network link
- Replicating with Data Pump transportable tablespaces
- Replicating with RMAN transportable tablespaces between the same operating system platform and between different operating system platforms
- Moving data with external tables

Understanding these manual methods lays the foundation for intelligently using RMAN's duplicate database functionality. You'll now better understand the advantages and disadvantages of each feature. You will be in a better position to architect replication solutions. The RMAN duplicate database feature is next discussed in detail in the Chapters 3, 4, and 5 in this book.

CHAPTER 3

Backup-Based Duplication

Backup-based duplication uses an RMAN backup of the target (source) database as its source to create the data files in the auxiliary (destination) environment. There are two types of RMAN backup-based duplication:

1. No connection to the target database (or a recovery catalog) is required. This is referred to as *targetless duplication*. This technique only requires a connection to the auxiliary database. Targetless duplication is available in Oracle 11g release 2 and higher.
2. In some types of backup-based duplication, a connection to both the target database (or a recovery catalog) and the auxiliary database is required.

The main focus of this chapter is targetless duplication, where there is no requirement to be connected to the target (source) database. In other words, you only need a connection to the auxiliary (destination) database. The big advantage to backup-based duplication is that if you work in an environment where it's not possible to have a simultaneous connection to both the target and the auxiliary database you can still duplicate a database provided you can copy an RMAN backup to the auxiliary database server (or provided the backup is on network-mounted storage readable from the auxiliary server). For example, in many environments, due to security rules, there is no network connectivity allowed from test environments to the production server.

The first section of this chapter outlines some basic troubleshooting techniques. If you're already familiar with basic troubleshooting, then proceed directly to the second major section of this chapter, which deals with targetless duplication. The last section of this chapter details a few backup-based duplication scenarios that require a connection to both the target database (and/or recovery catalog) and the auxiliary database.

Basic Troubleshooting

Before getting started with examples of duplicating databases, it's prudent to spend just a small amount of time going over some basic troubleshooting techniques, such as:

- Checking the syntax of an RMAN command
- Monitoring
- Logging RMAN output

Reviewing these techniques will save you a great deal of time when performing RMAN duplication operations. When you're having issues and can't get a command to run correctly, the first thing to do is check the syntax for accuracy. It may seem like a small thing, but one misplaced comma can cause hours of misplaced troubleshooting. If you're not familiar with these basic troubleshooting techniques, now would be a good time to spend just a few minutes reviewing the material in this section.

Checking Syntax

As you'll soon see, some of the RMAN *DUPLICATE* commands in coming examples can get quite lengthy, especially those using the *SPFILE* clause and specifying how to map different directory structures from the target to the auxiliary. With long *DUPLICATE* commands, it's easy to miss a small detail and attempt to run a command that isn't syntactically correct. In these situations, it's handy to check first (before actually running the RMAN command) to see if the syntax is accurate. Fortunately, RMAN has a built-in *CHECKSYNTAX* clause that allows you to do this. You can use this clause in a couple of different ways.

From the Command Line

One method for using the *CHECKSYNTAX* clause is from the operating system command line. To do so, initiate RMAN with the *CHECKSYNTAX* clause:

```
$ rman checksyntax
```

While connected to RMAN in this mode, any commands that you issue aren't executed; rather, RMAN only checks to see if the syntax is correct. For example

```
RMAN> duplicate database to TRG
backup location '/u01/rman/TRG'
nofilenamecheck;
```

If everything is correct, this line is displayed in the output:

```
The command has no syntax errors
```

The *CHECKSYNTAX* clause can be used for any RMAN command. Most RMAN commands aren't very long, like *BACKUP DATABASE*. However, for lengthy *DUPLICATE* commands it's prudent to first verify the syntax before attempting to run the command. This will eliminate a syntax error as being the cause of a problem.

From a Script

Another technique for invoking the *CHECKSYNTAX* functionality is to put the RMAN command in an operating system file and then use that file as input to the RMAN. For example, say you have a long RMAN command stored in a file named *cmd.rc*. You can quickly check for syntax issues like this:

```
$ rman checksyntax @cmd.rc
```

If successful, this message is displayed:

```
The cmdfile has no syntax errors
```

■ **Tip** The *CHECKSYNTAX* clause does not check to see if required directories on the auxiliary (destination) server exist. If required directories don't exist, the *DUPLICATE* command will fail and throw an error indicating there has been a problem.

For lengthy RMAN commands, the *CHECKSYNTAX* clause provides you a quick way to determine if the issue is related to syntax of if it's some other problem. Eliminating syntax-related issues quickly helps you focus on the root cause of the problem.

Monitoring Progress

Sometimes when duplicating a database invariably somebody will ask "Is it done yet?" This is especially true for large databases. In order to better report on RMAN operations, I recommend that before you start any job (*DUPLICATE*, *BACKUP*, *RESTORE*, and so on) you first set the *NLS_DATE_FORMAT* operating system variable, as this will provide RMAN timings down to the second:

```
$ export NLS_DATE_FORMAT='dd-mon-rrrr hh24:mi:ss'
```

Now RMAN will report down to the second when each operation took place. If you don't set the prior variable then RMAN only reports the day, month, and year portion of the time. If RMAN operations are common in your environment then consider setting the *NLS_DATE_FORMAT* in an operating system startup file so that the variable is consistently set when you log on to a server.

Next to be covered are several methods for monitoring progress. First up is the operating system approach.

Operating System Approach

When a duplication operation is taking place, you can navigate to the directory (or directories) that contain the auxiliary database data files and use Linux/UNIX operating system utilities such as *ls* or *du* to manually view which data files have been restored and/or are currently being restored. For instance:

```
$ cd /u01/dbfile/TRG
$ ls -altr
ls -altr
total 2597754
drwxr-x---   4 oracle   dba              4 Dec 19 21:01 ..
drwxr-x---   2 oracle   dba             10 Jan  2 10:07 .
-rw-r-----   1 oracle   dba       19185664 Jan  2 10:07 control01.ctl
-rw-r-----   1 oracle   dba       19185664 Jan  2 10:07 control02.ctl
-rw-r-----   1 oracle   dba      524296192 Jan  2 10:07 system01.dbf
-rw-r-----   1 oracle   dba      524296192 Jan  2 10:07 sysaux01.dbf
-rw-r-----   1 oracle   dba      209723392 Jan  2 10:07 undotbs01.dbf
...
```

This approach is simple but very effective. This will give you an idea as to where RMAN is in the duplication process. If your target (source) database has hundreds of data files and you only see a handful of data files that are in the auxiliary destination directory, then you know there is a bit more work to be done.

If you're in a Windows environment, you can view the file sizes from the Windows Explorer graphical tool, or use the DOS command-line *DIR/O:-S* command to view file sizes.

SQL Approach

The SQL*Plus script in this section will give you an idea of how long the duplication process has remaining. First, some background on how RMAN operates. RMAN will create at least one process, called a *channel*, when performing the duplication operation. You can also instruct RMAN to open more than one channel (see the section on "Parallelism" in Chapter 5 for details). Each channel has an associated database process that can be monitored through SQL*Plus. RMAN will open the channels, either on the

target database or the auxiliary database depending on the type of duplication you're performing. Here are some guidelines regarding where RMAN will open the channels:

- **When performing active duplication with the backup set format.** Most of the work is done by the auxiliary database channels and therefore you should run the script in this section on the auxiliary database to view progress.
- **When performing active duplication when image copies are specified.** The work is done by target database channels, therefore run the script in this section on the target database to view progress. This is important to remember when building a standby from an active database using image copies; in this situation you'll need to run the monitor script on the target database.
- **When performing targetless duplication.** All of the work is done on the auxiliary database and therefore the script should be run on the auxiliary database to view the progress of the duplication job.

Based on these listed rules, you'll have to run the following SQL script to monitor RMAN progress either on the target or on the auxiliary, depending on where RMAN opens the channels:

```
SET LINES 132
COL opname        FORM A30 HEAD "Oper."
COL pct_complete  FORM 99.99 HEAD "% Comp."
COL start_time    FORM A15 HEAD "Start|Time"
COL hours_running FORM 9999.99 HEAD "Hours|Running"
COL minutes_left  FORM 999999 HEAD "Minutes|Left"
COL est_comp_time FORM A15 HEAD "Est. Comp.|Time"
--
SELECT sid, serial#, opname,
ROUND(sofar/totalwork*100,2) AS pct_complete,
TO_CHAR(start_time,'dd-mon-yy hh24:mi') start_time,
(sysdate-start_time)*24 hours_running,
((sysdate-start_time)*24*60)/(sofar/totalwork) - (sysdate-start_time)*24*60
minutes_left,
TO_CHAR((sysdate-start_time)/(sofar/totalwork) + start_time,'dd-mon-yy
hh24:mi') est_comp_time
FROM  v$session_longops
WHERE opname LIKE 'RMAN%'
AND   opname NOT LIKE '%aggregate%'
AND   totalwork != 0
AND   sofar <> totalwork;
```

Here is some sample output that results when performing active duplication and running the script on the target database during an image copy–based duplication operation:

```
                                                    Start          Hours     Minutes Est. Comp.
 SID SERIAL#  Oper.                        % Comp.  Time           Running   Left    Time
---- -------  -------------------------- -------  -------------- --------- ------  --------------
  34   37827  RMAN: full datafile backup    5.60  31-dec-14 10:02      .01     11  31-dec-14 10:14
  51   10159  RMAN: full datafile backup   20.21  31-dec-14 10:02      .01      3  31-dec-14 10:05
```

The output doesn't quite fit on the page, but if you have a wide terminal this will clearly show when the operation started, how long it's been running, and how much time is remaining. The output also indicates two RMAN channels have been allocated on the target and that, as part of an image copy, RMAN is backing up the data files.

■ **Note** You can run the prior script to provide timing details on any type of RMAN operation, such as how long a backup has been running or how long a restore operation has remaining.

Capturing RMAN Output

The RMAN *DUPLICATE* command in particular can generate great volumes of output. Sometimes when troubleshooting it's handy to capture all of the output in a file that can be used later to identify issues. You can either use the RMAN logging feature or use an operating system utility to capture the output.

RMAN Logging

RMAN logging can be enabled a couple of different ways. When connected to RMAN you can specify a log file to capture the output of your activities:

```
RMAN> spool log to '/u01/log/backup.log';
RMAN> backup database;
RMAN> spool log off;
Spooling for log turned off
```

You should now have a file in the */u01/log* directory named *backup.log* that contains the output of the *BACKUP* command. Note that the directory */u01/log* must exist on the server for this to work. You'll have to modify this appropriately for your environment.

The other way to enable RMAN logging is from the command line:

```
$ rman target / log /u01/log/output.log
```

Now all output is captured for any subsequent RMAN commands in the */u01/log/output.log* file.

Script Command

The Linux/UNIX *script* command enables the recording of all output printed to your terminal to additionally be captured in an operating system file. For instance, before connecting to RMAN and running a *DUPLICATE* command, first do this:

```
$ script dup.log
Script started, file is dup.log
```

Now connect to RMAN and run the *DUPLICATE* command:

```
$ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo@shrek2:1521/TRG
RMAN> DUPLICATE TARGET DATABASE TO TRG
FROM ACTIVE DATABASE
USING COPY
NOFILENAMECHECK;
```

When finished, you can stop the script-capture process by pressing *Ctrl+D* or typing in *exit*, after which you should see this message:

```
Script done, file is dup.log
```

Now you can use an operating system utility such as *vi* or *notepad* to examine the contents of the script file. If you enlist the help of Oracle Support with an issue, a file such as this containing all the command output can be invaluable in troubleshooting problems.

Tee Command

The Linux/UNIX *tee* command is another useful way to record output displayed on the terminal into a file. Here's an example of using *tee*:

```
$ rman target / | tee rman.log
```

Now any commands that you run will be captured in the *rman.log* file. When you exit RMAN, the *rman.log* file closes.

RMAN Command Output View

The *V$RMAN_OUTPUT* view contains messages recently reported by RMAN. It is an in-memory view that can hold a maximum of 32,768 rows. Information in this view is cleared out when you stop and start your database. Here's a sample way to query this view:

```
set lines 132 pages 110
col output form a70
--
select sid, recid, output
from v$rman_output
order by recid;
```

Using *V$RMAN_OUTPUT* also has the advantage of being operating system agnostic. You can always query its output regardless of the operating system (Windows, Linux, Solaris, and so on).

Targetless Duplication

You can replicate a target (source) database using an RMAN backup when duplicating to an auxiliary (destination) database. In the scenarios described in this section, you are not required to connect to the target database or a recovery catalog when issuing the RMAN *DUPLICATE* command (hence the moniker targetless duplication). The basic idea here is to copy an RMAN backup to an auxiliary server (or storage that the auxiliary server has read access to) and create the auxiliary database directly from the backup. This is a simple and powerful technique for replicating a database. It is especially applicable where there's no direct network connection between the target database and the auxiliary.

Next, several targetless duplication scenarios are described. Let's get started with the simplest scenario, one in which the directory structure and database name are the same from the target to the auxiliary environment.

Directory Structure and Database Name Remain Identical

It is easiest to use the RMAN *DUPLICATE* command when you have the following scenario:

- Target (source) and auxiliary (destination) servers have the same directory structure, meaning that the directory locations on the target for data files, control files, and online redo logs are identical to the directory locations on the auxiliary server.
- Target and auxiliary database names are the same, meaning you don't require the name of the auxiliary database to be different from the target.

In this situation, the basic idea is that you copy the RMAN backup files from the target (source) server to the auxiliary (destination) server and then issue the *DUPLICATE* command to create a copy of the target database on the auxiliary server. As shown in Figure 3-1, there are five steps required for this scenario. Notice this configuration doesn't

require a listener or Oracle Net connectivity. The auxiliary database simply requires read access to an RMAN backup of the target database. In this situation the RMAN backup is copied to a directory on the auxiliary server.

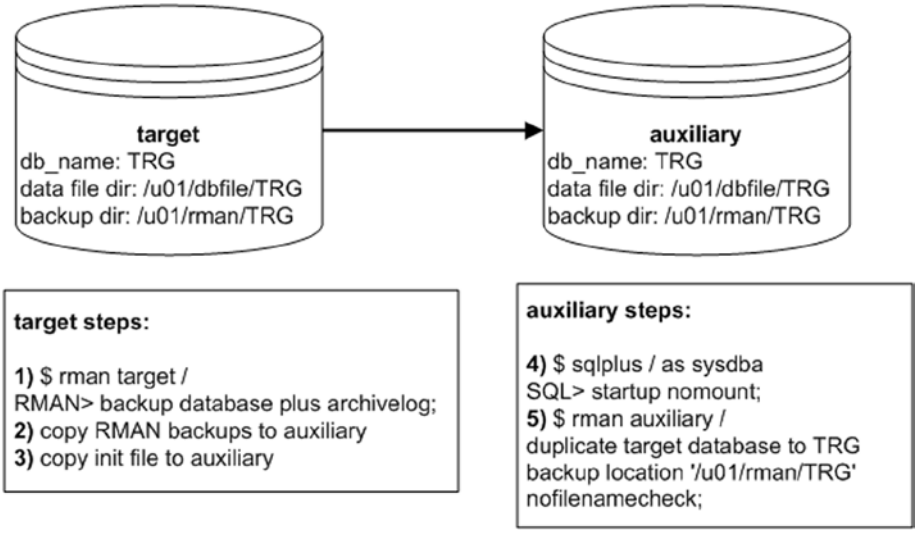

Figure 3-1. *Same name and directory structure using targetless duplication*

The first three steps are executed on the target server, and the last two steps are executed on the auxiliary server.

1. On the target server, connect to RMAN and back up the target (source) database plus archivelog:

   ```
   $ rman target /
   RMAN> backup database plus archivelog;
   ```

2. On the target server, copy the target RMAN backups to the auxiliary (destination) server. First locate the RMAN backups and then copy them to the destination server:

   ```
   RMAN> list backup;
   ```

Here's a partial listing of the output for my target database showing that the RMAN backup pieces (physical backup files) are in the */u01/rman/TRG* directory:

```
Piece Name: /u01/rman/TRG/TRGrman1_07pqj1um_1_1.bk
```

Next I use the Linux/UNIX *scp* command to copy the RMAN backups from the source server to the destination server (you'll have to modify this per the location of your backups on the source and destination servers):

```
$ cd /u01/rman/TRG
$ scp *.bk oracle@shrek2:/u01/rman/TRG
```

As mentioned at the beginning of the chapter, another method to make the RMAN backup available to the auxiliary server would be to place the backups on storage that is readable from the auxiliary host. If this option is available it would save you from having to copy the backups (which takes time and storage).

3. On the target server, copy the target *init.ora* to the auxiliary server (you'll have to modify this per the location of your backups on the source and destination servers). You can use either a server parameter file (SPFILE) or an *init.ora* file; here I'm using an SPFILE:

   ```
   $ cd $ORACLE_HOME/dbs
   $ scp spfileTRG.ora oracle@shrek2:$ORACLE_HOME/dbs
   ```

4. Now log on to the auxiliary server, connect to SQL*Plus, and start up the auxiliary database in nomount mode:

   ```
   $ sqlplus / as sysdba
   SQL> startup nomount;
   SQL> exit;
   ```

5. On the auxiliary server, connect to the auxiliary instance via RMAN and issue the *DUPLICATE* command:

   ```
   $ rman auxiliary /
   RMAN> duplicate database to TRG
   backup location '/u01/rman/TRG'
   nofilenamecheck;
   ```

Here's a very small snippet of the large amount of output for this operation:

```
Starting Duplicate Db at...
contents of Memory Script:
...
Finished Duplicate Db...
```

When finished, you should have a database restored on the destination server that is an identical copy of the source database.

One important note: In the *DUPLICATE* command the *NOFILENAMECHECK* was included. When the target database and auxiliary database are on different servers, but have the same directory structure, then always include the *NOFILENAMECHECK* clause. If you don't include *NOFILENAMECHECK*, then you'll receive the following error:

```
RMAN-05001: auxiliary file name ... conflicts with a
file used by the target database
```

The *NOFILENAMECHECK* instructs RMAN not to check that the data file names are identical from the target to the auxiliary. If your target and auxiliary are on the same server, then you should not include the *NOFILENAMECHECK*, as you want RMAN to ensure that you don't attempt to restore an auxiliary data file over the top of a live target database data file.

Directory Structure Identical and Database Name Different

In this scenario the target (source) server and auxiliary (destination) server have the same directory structures. However, you want to rename the database as part of duplicating the target to the auxiliary. As shown in Figure 3-2, there are six steps required for this scenario.

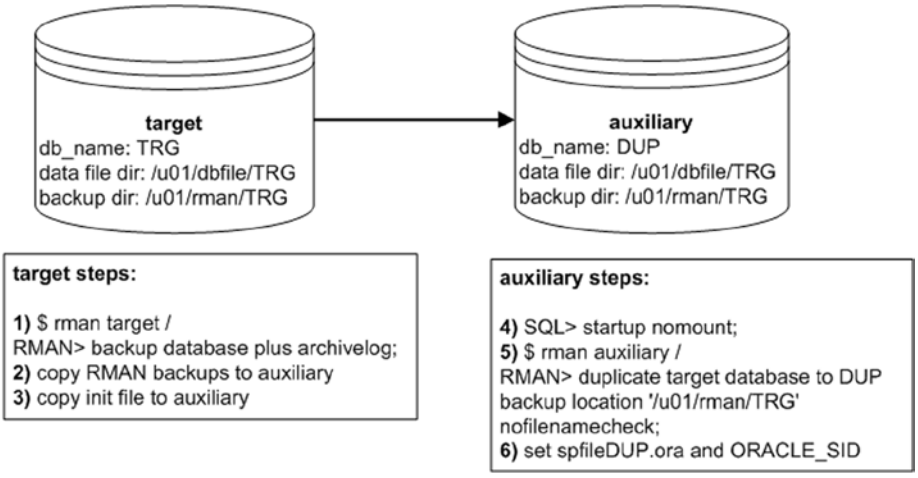

Figure 3-2. *Duplicating and changing the database name*

The first three steps are executed on the target server, and the last several steps are executed on the auxiliary server:

1. On the target server, connect to RMAN and back up the target (source) database plus archivelog:

```
$ rman target /
RMAN> backup database plus archivelog;
```

2. On the target server, copy the target RMAN backups to the auxiliary (destination) server. Start by verifying the location of the backups:

```
RMAN> list backup;
```

Here's a partial listing of the output for my target database that shows the RMAN backup pieces (physical backup files) are in the */u01/rman/TRG* directory:

```
Piece Name: /u01/rman/TRG/TRGrman1_0fpqj2f9_1_1.bk
```

Next use the Linux/UNIX *scp* command to copy the RMAN backups from the source server to the destination server (you'll have to modify this per the location of your backups on the source and destination servers):

```
$ cd /u01/rman/TRG
$ scp *.bk oracle@shrek2:/u01/rman/TRG
```

3. On the target server, copy the target initialization file to the auxiliary server (you'll have to modify this per the location of your backups on the source and destination servers). For this scenario I'm using a text-based *init.ora* file:

```
$ cd $ORACLE_HOME/dbs
$ scp initTRG.ora oracle@shrek2:$ORACLE_HOME/dbs
```

If your source database uses an SPFILE, then you can create a text-based *init.ora* file from SQL*Plus, as follows:

```
SQL> create pfile from spfile;
```

This command will place a text-based initialization file with the name of *init<SID>.ora* in the *ORACLE_HOME/dbs* directory. If you don't want the text-based file to be placed in that directory you can override the default behavior as follows:

```
SQL> create pfile='/tmp/initTRG.ora' from spfile;
```

4. Now log on to the auxiliary server, connect to SQL*Plus, and start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

5. On the auxiliary server, connect to the auxiliary instance via RMAN and issue the *DUPLICATE* command. Notice that the database is being duplicated and given the new name of *DUP*:

```
$ rman auxiliary /
RMAN> duplicate database to DUP
backup location '/u01/rman/TRG'
nofilenamecheck;
```

Here's a very small snippet of the large amount of output for this operation:

```
Starting Duplicate Db at...
contents of Memory Script:
...
Finished Duplicate Db...
```

6. When finished, the database name in this example is *DUP*, but it still has an instance name of *TRG*. To set the instance name to *DUP*, shut down the auxiliary database:

```
RMAN> shutdown immediate;
RMAN> exit;
```

In this scenario, RMAN will automatically create an SPFILE for you with the name of *spfileTRG.ora*. You must rename that file:

```
$ cd $ORACLE_HOME/dbs
$ mv spfileTRG.ora spfileDUP.ora
```

To complete this operation you need to set your *ORACLE_SID* to reflect the new database name of *DUP*:

```
$ export ORACLE_SID=DUP
$ sqlplus / as sysdba
SQL> startup;
```

■ **Tip** See MOS note 874352.1 for additional details regarding targetless duplication.

Directory Structures and Database Names Different, Using SPFILE Clause

In this scenario the directory structure is different from the target (source) host to the auxiliary (destination) host, and the auxiliary database will be renamed (to be different from the target). The *SPFILE* clause will be used to facilitate this operation. When performing targetless duplication, in order to use the *SPFILE* clause, two conditions apply:

- Ensure that the target database is using an SPFILE when the backup is created, otherwise you'll receive this error when duplicating to the auxiliary:

  ```
  RMAN-05569: SPFILE backup not found in /u01/rman/TRG
  ```

- The auxiliary database must be started with an *init.ora* file (and not an SPFILE), otherwise you'll receive this error when duplicating:

  ```
  RMAN-05537: DUPLICATE without TARGET connection when
  auxiliary instance is
  started with spfile cannot use SPFILE clause
  ```

Figure 3-3 depicts the steps used in this scenario.

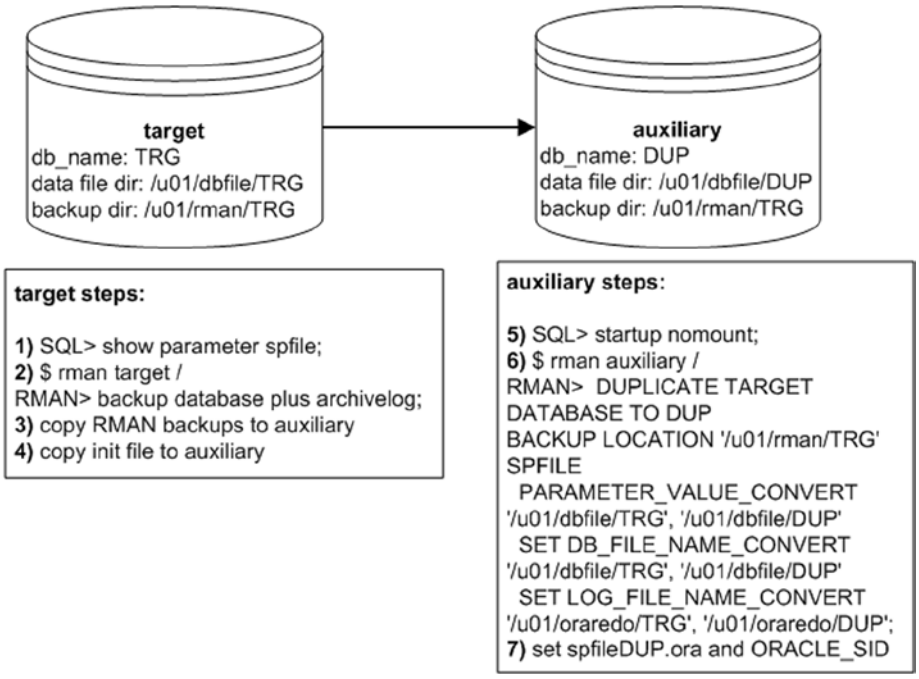

Figure 3-3. *Duplicating with different directory structure and database name*

1. The first step is to verify the target database is using an SPFILE-based initialization file:

```
SQL> select value from v$parameter where name='spfile';
VALUE
-----------------------------------------------------------
/orahome/app/oracle/product/12.1.0.2/db_1/dbs/spfileTRG.ora
```

   You can also verify the target database is using an SPFILE via:

```
SQL> show parameter spfile;
```

2. On the target server, connect to RMAN and back up the target (source) database plus archivelog. Additionally, a full backup of the database will automatically include the SPFILE. The SPFILE must be in the backup when using the *SPFILE* clause in the *DUPLICATE* command (seen in a subsequent step of this scenario):

```
$ rman target /
RMAN> backup database plus archivelog;
```

3. On the target server, locate the RMAN backups and copy them to the destination server:

```
RMAN> list backup;
```

Here's a partial listing of the output for my target database showing that the RMAN backup pieces (physical backup files) are in the */u01/rman/TRG* directory. Ensure that the output shows an SPFILE was included:

```
Piece Name: /u01/rman/TRG/TRGrman1_0spq30vi_1_1.bk
Piece Name: /u01/rman/TRG/TRGrman2_umpu0r0e_1_1.bk
SPFILE Included: Modification time...
```

Next use the Linux/UNIX *scp* command to copy the RMAN backups from the source server to the destination server (you'll have to modify this per the location of your backups on the source and destination servers):

```
$ cd /u01/rman/TRG
$ scp *.bk oracle@shrek2:/u01/rman/TRG
```

4. On the target server, copy the target initialization file to the auxiliary server. In this scenario you must use an *init.ora* file when starting the auxiliary database, otherwise RMAN will throw the following error:

```
RMAN-05537, DUPLICATE without TARGET connection when auxiliary
instance is started with spfile cannot use SPFILE clause.
```

Create a text-based *init.ora* file and then copy it to the auxiliary server:

```
$ sqlplus / as sysdba
SQL> create pfile from spfile;
SQL> exit;
$ cd $ORACLE_HOME/dbs
$ scp initTRG.ora oracle@shrek2:$ORACLE_HOME/dbs
```

5. Now log on to the auxiliary server, connect to SQL*Plus, and start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

6. Connect to the auxiliary database and issue the *DUPLICATE* command. Notice how the *DUPLICATE* command specifies the new locations for control files (*PARAMETER_VALUE_CONVERT*), data files (*DB_FILE_NAME_CONVERT*), and online redo log files (*LOG_FILE_NAME_CONVERT*). You'll have to modify this statement to reflect the directory structures in your environment. These directories must exist on the auxiliary server (RMAN doesn't create the directories for you):

```
$ rman auxiliary /
RMAN> DUPLICATE TARGET DATABASE TO DUP
BACKUP LOCATION '/u01/rman/TRG'
SPFILE
  PARAMETER_VALUE_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET DB_FILE_NAME_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET LOG_FILE_NAME_CONVERT
'/u01/oraredo/TRG', '/u01/oraredo/DUP';
```

When finished you should have a database with the new name of *DUP*, but the instance is currently running with the old name of *TRG*. You should also have an SPFILE that is named *spfileTRG.ora*.

7. To start your database with the new name of *DUP*, shut down the database, rename the SPFILE, export your *ORACLE_SID* to reflect the new instance name, and then restart your database:

```
RMAN> shutdown immediate;
RMAN> exit
$ cd $ORACLE_HOME/dbs
$ mv spfileTRG.ora spfileDUP.ora
```

```
$ export ORACLE_SID=DUP
$ sqlplus / as sysdba
SQL> startup;
```

Keep in mind when using the *SPFILE* clause that you can specify other initialization parameters for which you require different values from the target to the auxiliary. For example, this next bit of code also sets the values of *MEMORY_TARGET*, *MEMORY_MAX_TARGET*, and *SHARED_POOL_SIZE*. For instance:

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
BACKUP LOCATION '/u01/rman/TRG'
SPFILE
  PARAMETER_VALUE_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET DB_FILE_NAME_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET LOG_FILE_NAME_CONVERT
'/u01/oraredo/TRG', '/u01/oraredo/DUP'
  SET MEMORY_TARGET '800M'
  SET MEMORY_MAX_TARGET '800M'
  SET SHARED_POOL_SIZE '100M';
```

After the duplication process is complete, you can verify the memory settings from SQL*Plus:

```
SQL> show parameter shared_pool_size;
NAME              TYPE        VALUE
----------------- ----------- -------
shared_pool_size big integer 100M
```

Directory Structures and Database Names Different, Not Using SPFILE

In this scenario, your target (source) and auxiliary (destination) servers have different directory structures, and the auxiliary database is renamed. Additionally, the target database does not use an SPFILE, so you don't have the option of using the *SPFILE* clause of the *DUPLICATE* command.

1. On the target server, connect to RMAN and back up the target (source) database:

   ```
   $ rman target /
   RMAN> backup database plus archivelog;
   ```

2. On the target server, locate the RMAN backups and copy them to the destination server:

   ```
   RMAN> list backup;
   ```

Here's a partial listing of the output for my target database showing that the RMAN backup pieces (physical backup files) are in the */u01/rman/TRG* directory:

```
Piece Name: /u01/rman/TRG/TRGrman1_0fpqj2f9_1_1.bk
```

On the target server, use the Linux/UNIX *scp* command to copy the RMAN backups from the source server to the destination server (you'll have to modify this per the location of your backups on the source and destination servers):

```
$ cd /u01/rman/TRG
$ scp *.bk oracle@shrek2:/u01/rman/TRG
```

3. On the target server, copy the target database *init.ora* file to the auxiliary server. Notice that the *init.ora* file is renamed during the copying to *initDUP.ora*:

```
$ cd $ORACLE_HOME/dbs
$ scp initTRG.ora oracle@shrek2:$ORACLE_HOME/dbs/initDUP.ora
```

4. Now on the auxiliary database server, modify the *initDUP.ora* file to reflect the new directory structure. For this example, I've modified the *CONTROL_FILES* and *DB_NAME* parameters as follows:

```
control_files='/u01/dbfile/DUP/control01.ctl',
'/u01/dbfile/DUP/control02.ctl'
db_name='DUP'
```

5. Next, export the *ORACLE_SID* to reflect the new database name:

```
$ export ORACLE_SID=DUP
```

Now start up the auxiliary in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

6. Connect to the auxiliary database via RMAN and issue the *DUPLICATE* command:

```
$ rman auxiliary /
RMAN> DUPLICATE TARGET DATABASE TO DUP
BACKUP LOCATION '/u01/rman/TRG'
  DB_FILE_NAME_CONVERT '/u01/dbfile/TRG','/u01/dbfile/DUP'
                       ,'/u02/dbfile/TRG','/u02/dbfile/DUP'
                       ,'/u03/dbfile/TRG','/u03/dbfile/DUP'
LOGFILE GROUP 1('/u01/oraredo/DUP/redo01a.rdo') SIZE 50m,
        GROUP 2('/u01/oraredo/DUP/redo02a.rdo') SIZE 50m;
```

When finished you should have a fully functioning copy of the target database on the auxiliary server. Note that even though the *SPFILE* clause wasn't specified in this scenario, RMAN will still create an SPFILE for the auxiliary database. As a result, you may see this in the output:

```
Cannot remove created server parameter file
```

This is just an informational message and nothing to worry about.

Transforming Directory Names via Initialization File

Instead of specifying in the *DUPLICATE* command the transformation of the target directory structure to the auxiliary directory structure, you can place the necessary transformation in the auxiliary initialization file directly. For example, the following modifications to the auxiliary initialization file (*initDUP.ora* in this example) instruct RMAN where to place the control files (*CONTROL_FILES*), how to transform data file names (*DB_FILE_NAME_CONVERT*), and how to transform online redo log file names (*LOG_FILE_NAME_CONVERT*):

```
db_name=DUP
#
control_files='/u01/dbfile/DUP/control01.ctl',
'/u02/dbfile/DUP/control02.ctl'
#
db_file_name_convert= ('/u01/dbfile/TRG','/u01/dbfile/DUP','/u02/dbfile/TRG',
'/u02/dbfile/DUP',
'/u03/dbfile/TRG','/u03/dbfile/DUP')
#
log_file_name_convert=('/u01/oraredo/TRG','/u01/oraredo/DUP',
'/u02/oraredo/TRG','/u02/oraredo/DUP')
```

And now start up your database in nomount mode:

```
$ export ORACLE_SID=DUP
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

Next, use RMAN to connect to the auxiliary database and issue the *DUPLICATE* command:

```
$ rman auxiliary /
RMAN> DUPLICATE TARGET DATABASE TO DUP
BACKUP LOCATION '/u01/rman/TRG';
```

When finished, you should see this at the bottom of the lengthy output:

```
Finished Duplicate Db at...
```

Now you should stop and start your duplicated database and inspect the *alert.log* file. You may see an error such as this:

```
ORA-01110: data file 201: '/u01/dbfile/DUP/temp01.dbf'
```

To resolve this either move or remove the existing temporary tablespace temp file using operating system commands:

```
$ mv /u01/dbfile/DUP/temp01.dbf /u01/dbfile/DUP/tempo01.old.dbf
```

Then stop and start the database. When starting, Oracle will detect that the temporary tablespace temp file is missing and recreate it.

Transforming Directory Names using SET NEWNAME

Another technique for instructing RMAN to transform directory names is with the *SET NEWNAME* command. This command must be encapsulated within an RMAN *RUN{}* block. Before performing this operation, first verify your target database data file numbers and corresponding names:

```
$ rman target /
RMAN> report schema;
```

In the following output, take note of the file number and data file name; this will provide the mapping used subsequently with the *SET NEWNAME* command:

```
File Size(MB) Tablespace  RB segs Datafile Name
---- -------- ----------  ------- ----------------------------
1    500      SYSTEM      YES     /u01/dbfile/TRG/system01.dbf
2    500      SYSAUX      NO      /u01/dbfile/TRG/sysaux01.dbf
3    200      UNDOTBS1    YES     /u01/dbfile/TRG/undotbs01.dbf
4    15       USERS       NO      /u01/dbfile/TRG/users01.dbf
5    10       REPDATA     NO      /u01/dbfile/TRG/repdata.dbf
6    10       REPIDX      NO      /u01/dbfile/TRG/repidx.dbf

List of Temporary Files
=======================
File Size(MB) Tablespace  Maxsize(MB) Tempfile Name
---- -------- ----------  ----------  -------------------------
1    500      TEMP        500         /u01/dbfile/TRG/temp01.dbf
```

On the auxiliary host, set your *ORACLE_SID* to the name of the new database:

```
$ export ORACLE_SID=DUP
```

Next, modify the initialization file so that it reflects the new name of the database and the directories for the control files (in this example the initialization file name is *initDUP.ora*):

```
db_name=DUP
#
control_files='/u01/dbfile/DUP/control01.ctl','/u02/dbfile/DUP/control02.ctl'
```

And now start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

Next, use RMAN to connect to the auxiliary database and issue the *SET NEWNAME* and *DUPLICATE* commands with a *RUN{}* block:

```
$ rman auxiliary /
```

Now run the following code. Notice the new directory location has been specified for each data file (*TRG* has been changed to *DUP*):

```
RMAN> RUN
{
SET NEWNAME FOR DATAFILE 1 TO '/u01/dbfile/DUP/system01.dbf';
SET NEWNAME FOR DATAFILE 2 TO '/u01/dbfile/DUP/undotbs01.dbf';
SET NEWNAME FOR DATAFILE 3 TO '/u01/dbfile/DUP/sysaux01.dbf';
SET NEWNAME FOR DATAFILE 4 TO '/u01/dbfile/DUP/users01.dbf';
SET NEWNAME FOR DATAFILE 5 TO '/u01/dbfile/DUP/repdata.dbf';
SET NEWNAME FOR DATAFILE 6 TO '/u01/dbfile/DUP/repidx.dbf';
SET NEWNAME FOR TEMPFILE 1 TO '/u01/dbfile/DUP/temp01.dbf';
DUPLICATE TARGET DATABASE TO DUP
BACKUP LOCATION '/u01/rman/TRG'
LOGFILE GROUP 1 ('/u01/oraredo/DUP/redo01.rdo') SIZE 50M,
        GROUP 2 ('/u01/oraredo/DUP/redo02.rdo') SIZE 50M;
}
```

When finished you should see the following line at the bottom of the lengthy output:

```
Finished Duplicate Db at...
```

Shell Scripting the Duplication Process

Oftentimes you'll have the requirement to automate the duplication process. For example, in one environment at work the development team regularly requests that I refresh a development copy of the production database to the test environment. I use the Linux/UNIX *cron* utility to schedule the following tasks:

- Copy the latest RMAN target database backup from production to the auxiliary test server
- Copy an initialization file from the target to the auxiliary server
- Run a Bash shell script (on the auxiliary server) to refresh the database

Here are the contents of the Bash shell script that automate the duplication of the target database:

```
#!/bin/bash
#---------------------------------
export ORACLE_SID=TRG
#
sqlplus -s /nolog <<EOF
connect / as sysdba;
shutdown immediate;
startup nomount;
exit;
EOF
#---------------------------------
rman nocatalog <<EOF
connect auxiliary /
duplicate database to TRG
backup location '/u01/rman/TRG'
nofilenamecheck;
exit;
EOF
#---------------------------------
exit 0
```

The shell script needs to exist on the auxiliary (destination) database server. Ensure you make the shell script executable. Assuming the shell script name is *dup.bsh*, here's an example of making the shell script executable:

```
$ chmod +x dup.bsh
```

And now you should be able to run the script:

```
$ dup.bsh
```

When RMAN is finished duplicating the database, you should see the following line:

```
Recovery Manager complete.
```

You should now have a database that has been duplicated from the target database RMAN backups.

Sometimes with large databases it can take a while for the shell script to complete. When executing the shell script from the command line, it's useful to use the Linux/UNIX *nohup* command. This will allow the shell script to run in the background. Sometimes it's advantageous to run a script in the background, as that will prevent the shell script from being terminated if the server is configured to automatically terminate what appears to be a process with no activity. Here's an example of using *nohup*:

```
$ nohup dup.bsh &
```

Now you can monitor the progress of the duplication job via the Linux/UNIX *tail* command:

```
$ tail -f nohup.out
```

The *-f* switch instructs the *tail* command to continuously display the last several lines of the output on your terminal. To exit the continuous tailing process, enter a *Ctrl+C*.

Duplicating and Stopping Recovery at a Specific Time

It's possible to specify a specific point-in-time recovery when duplicating a database. You may want to do this because you want the restoration point to stop at a specific point (like a baseline that you've established for testing). First, start up the auxiliary (destination) database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

Next, connect via RMAN to the auxiliary:

```
$ rman auxiliary /
```

This next bit of code specifies that the *DUPLICATE* command should apply all transactions that occurred before the specified point in time:

```
RMAN> DUPLICATE TARGET DATABASE TO TRG
BACKUP LOCATION '/u01/rman/TRG'
db_file_name_convert '/u01/dbfile/TRG','/u01/dbfile/DUP'
                     ,'/u02/dbfile/TRG','/u02/dbfile/DUP'
                     ,'/u03/dbfile/TRG','/u03/dbfile/DUP'
```

```
logfile group 1
  ('/u01/oraredo/DUP/redo01a.rdo',
   '/u02/oraredo/DUP/redo01b.rdo') size 50m,
group 2
('/u01/oraredo/DUP/redo02a.rdo',
 '/u02/oraredo/DUP/redo02b.rdo') size 50m
UNTIL TIME "TO_DATE('16-dec-2014 09:00:00', 'dd-mon-rrrr hh24:mi:ss')";
```

■ **Tip** It's best to always specify the date format with the *TO_DATE* function. This eliminates any ambiguity as to what date is being utilized.

Restarting Duplication

One handy aspect of RMAN is that if you're restoring a large database and there's some sort of failure (e.g., power outage) that causes the restore to abort, when you restart the restore operation RMAN will not restore files that have already been successfully restored. RMAN checks for files in the expected location and expected information in the data file header and if already present, RMAN will not restore those files. This is known as *restore optimization*. For operations initiated with the *RESTORE* command, you can override restore optimization with the *FORCE* option. This forces RMAN to restore a file even if it exists in the expected location.

The restore optimization feature also applies to RMAN database duplication. If the duplication job unexpectedly aborts during the replication process you can simply rerun the *DUPLICATE* command and RMAN will not restore data files that were previously successfully restored. Thus, if you had a *DUPLICATE* job that was 90% complete, and the server crashed, when it comes back online, you can rerun the *DUPLICATE* command and RMAN will restore the remaining 10% of the data files.

When rerunning the *DUPLICATE* command, if RMAN detects there are data files that have already been restored, it will display a message similar to this in the output:

```
skipping datafile 1; already restored to file /u01/dbfile/TRG/system01.dbf
skipping datafile 4; already restored to file /u01/dbfile/TRG/users01.dbf
```

One caveat is that unlike operations initiated with the *RESTORE* command, with the *DUPLCIATE* command there is no way to force RMAN to restore files that have previously been successfully restored. If you need to force RMAN to restore a file, then first delete it.

Restricting Access after Duplication

By default, when you duplicate a database, as the last step RMAN will open the database. You may not want this behavior if you don't immediately want the duplicated database to be available for use. For example, you may first want to verify the duplication was

successful before you open the database for use. In this situation use the *NOOPEN* clause. For example, first connect to the auxiliary database:

```
$ rman auxiliary /
```

Then issue the *DUPLICATE* command with *NOOPEN*:

```
RMAN> duplicate database to TRG
noopen
backup location '/u01/rman/TRG'
nofilenamecheck;
```

When finished, you have a duplicate of the target database, but the database is placed in mount mode:

```
SQL> select open_mode from v$database;

OPEN_MODE
--------------------
MOUNTED
```

When you're ready you can open the database for use. Notice that you must use the *RESETLOGS* clause when you open the database:

```
SQL> alter database open resetlogs;
```

Another option you have is to have RMAN open the database in restricted mode. In this way, only users with the restricted session privilege (like users with the DBA role assigned to them) can connect to the database. First, connect to the auxiliary database:

```
$ rman auxiliary /
```

Then issue the *DUPLICATE* command with *OPEN RESTRICTED* clause:

```
RMAN> duplicate database to TRG
open restricted
backup location '/u01/rman/TRG'
nofilenamecheck;
```

In this mode, RMAN issues an *ALTER SYSTEM ENABLE RESTRICTED SESSION* command before opening the database. You can verify that logins are restricted via:

```
RMAN> select status, logins from v$instance;

STATUS       LOGINS
------------ ----------
OPEN         RESTRICTED
```

When desired you can enable normal logons via:

```
RMAN> alter system disable restricted session;
```

■ **Tip** Starting with Oracle 12c, you can run SQL statements directly from within RMAN without having to specify the SQL clause.

Scenarios Requiring Connections to Target

All of the previous examples in this chapter have only required a connection to the auxiliary database to perform backup-based duplication. Having said that, there are some types of backup-based duplication scenarios that require a connection to both the target database and the auxiliary database. An example of this is replicating and stopping the restoration process at a specific log sequence number or at a restore point. If you attempt to restore to a sequence number or restore point without a connection to both the target and the auxiliary database, RMAN will throw an *RMAN-05542* error indicating that only *UNTIL TIME* can be used with *DUPLICATE* without a target (and/or recovery catalog) connection. Examples of sequence and restore-point duplication are detailed in the following sections.

UNTIL Sequence

Before performing a log sequence–based recovery, verify which archive redo logs are included in the backup and determine which archive redo log you want to recover up to (but not including). You can verify the archive redo logs included in a backup by issuing the following on the target database:

```
RMAN> list backup of archivelog all;
```

Here is some sample output:

```
List of Archived Logs in backup set 15
  Thrd Seq     Low SCN    Low Time  Next SCN   Next Time
  ---- ------- ---------- --------- ---------- ---------
  1    28      351977     19-DEC-14 352064     19-DEC-14
```

To specify a log sequence number at which the restoration process should stop you must be connected to both the target (source) and the auxiliary (destination) databases. If you attempt to duplicate to a log sequence number with just a connection to the auxiliary database you'll receive the following error message:

```
RMAN-05542: Only UNTIL TIME can be used with DUPLICATE
without TARGET and CATALOG connections
```

In this scenario, on the auxiliary server initiate a connection to both the target database and the auxiliary database. This requires that Oracle Net connectivity exists between the two servers (see Chapter 6 for details on Oracle Net):

```
$ rman target sys/foo@TRG auxiliary sys/foo
```

Next, issue the *DUPLICATE* command and specify a log sequence to restore up to (but not including). This example restores up to but not including the specified log sequence number:

```
DUPLICATE TARGET DATABASE TO TRG
BACKUP LOCATION '/u01/rman/TRG'
db_file_name_convert '/u01/dbfile/TRG','/u01/dbfile/DUP'
                     ,'/u02/dbfile/TRG','/u02/dbfile/DUP'
                     ,'/u03/dbfile/TRG','/u03/dbfile/DUP'
logfile group 1
  ('/u01/oraredo/DUP/redo01a.rdo',
   '/u02/oraredo/DUP/redo01b.rdo') size 50m,
group 2
('/u01/oraredo/DUP/redo02a.rdo',
 '/u02/oraredo/DUP/redo02b.rdo') size 50m
UNTIL SEQUENCE 28;
```

After the job is finished, you should see this at the bottom of the output:

```
Finished Duplicate Db at...
```

If you have a recovery catalog in place, it's also possible to restore until a sequence while connected to the recovery catalog and the auxiliary database. For example:

```
$ rman auxiliary / catalog rcat/foo@cat
```

Next, issue the *DUPLICATE* command. Notice that the syntax here does not use the keyword *TARGET*:

```
RMAN> DUPLICATE DATABASE TRG TO TRG
BACKUP LOCATION '/u01/rman/TRG'
db_file_name_convert '/u01/dbfile/TRG','/u01/dbfile/DUP'
                     ,'/u02/dbfile/TRG','/u02/dbfile/DUP'
                     ,'/u03/dbfile/TRG','/u03/dbfile/DUP'
logfile group 1
  ('/u01/oraredo/DUP/redo01a.rdo',
   '/u02/oraredo/DUP/redo01b.rdo') size 50m,
group 2
('/u01/oraredo/DUP/redo02a.rdo',
 '/u02/oraredo/DUP/redo02b.rdo') size 50m
UNTIL SEQUENCE 28;
```

The key here is that the *UNTIL* clause requires a connection to the target (or recovery catalog) database along with a connection to the auxiliary database.

UNTIL Restore Point

This example uses a restore point to recover to a point in time. A restore point is a pointer to a system change number (SCN). Recall that an SCN is an internal counter that Oracle uses to assign a sequential number to every change that occurs in the database.

1. First, on the target (source) database, take an RMAN backup as follows:

```
RMAN> backup database
include current controlfile
plus archivelog;
```

2. Next, create a restore point on the target database.

```
SQL> create restore point my_rp;
```

3. Now back up all of the archive redo logs again, which will ensure that you have any archive redo logs required to restore up to the SCN recorded by the restore point:

```
RMAN> backup archivelog all;
```

4. Then use the Linux/UNIX *scp* command to copy the RMAN backups from the target server to the auxiliary server (you'll have to modify this per the location of your backups on the source and destination servers). This is initiated from the target server:

```
$ cd /u01/rman/TRG
$ scp *.bk oracle@shrek2:/u01/rman/TRG
```

5. Copy the initialization file from the target to the auxiliary. You can use either an SPFILE or a text-based *init.ora* file. Here's an example of using the Linux/UNIX *scp* command for my environment (initiated from the target server):

```
$ cd $ORACLE_HOME/dbs
$ scp $ORACLE_HOME/dbs/initTRG.ora oracle@shrek2:$ORACLE_HOME/dbs
```

6. Then on the auxiliary server, start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
```

7. On the auxiliary server, connect to the target and the auxiliary databases. When using an RMAN backup as the source and performing an *UNTIL SEQUENCE* duplication, a connection to both the target and the auxiliary is required:

```
$ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo
```

This connection can be made from either the target, the auxiliary, or a remote client.

8. Next issue the *DUPLICATE* command specifying a restore point. In this example the restore point is my *MY_RP*:

```
RMAN> DUPLICATE TARGET DATABASE TO TRG
BACKUP LOCATION '/u01/rman/TRG'
UNTIL RESTORE POINT my_rp
NOFILENAMECHECK;
```

At this point you may see the following errors:

```
RMAN-03002: failure of Duplicate Db command at ...
RMAN-05501: aborting duplication of target database
RMAN-03015: error occurred in stored script Memory Script
RMAN-06026: some targets not found - aborting restore
RMAN-06024: no backup or copy of the control file found to restore
```

This output indicates that RMAN can't retrieve the control file required or archive redo logs required from the backups. For example, if you create a restore point before you take a backup of the control file and data files, RMAN can't find a control file to restore that is prior to when the restore point was created.

In this situation, ensure that you have taken the RMAN backup before you created the restore point, and also have backed up any archive redo logs required to restore up to the SCN recorded in the restore point.

To further troubleshoot this issue, run the following on the target database:

```
RMAN> run {
set until restore point my_rp;
restore controlfile preview;
}
```

Inspect the output of the previous command. Ensure that the backups required to restore the data files and archived redo logs exist on the auxiliary server.

■ **Tip** See MOS note 1543996.1 for more details on the *RMAN-06024* error.

Summary

This chapter detailed using an RMAN backup as the source for the duplication operation. There are two types of backup-based duplication:

- Targetless duplication, which requires no connection to the target database
- Non-targetless duplication, which does require a target connection (and/or recovery catalog)

Most of the chapter discussed various targetless duplication scenarios. This is the simplest method for duplicating a database. This type of duplication relies only on an RMAN backup for the source of the database being copied. There is no need to connect to the target and/or a recovery catalog while performing targetless duplication.

This type of duplication is especially handy in environments where it's not possible to connect to the target and the auxiliary at the same time. This could be due to security requirements, and therefore no direct Oracle Net connection is possible between the target and the auxiliary.

CHAPTER 4

Active Duplication

The previous chapter discussed duplicating from an RMAN backup. This chapter walks you through duplicating directly from an open target database. This is referred to as *active duplication* because the source is a live database. You don't need an RMAN backup when using active duplication. Rather, RMAN uses the live data files and control files as the source when copying the target (source) to the auxiliary (destination). When actively duplicating, RMAN copies the files over the network in either:

- RMAN backup set format
- Image copy format

When using the backup set format, RMAN creates the equivalent of a backup set in memory and transfers the backup set across the network to the auxiliary database, where it restores and recovers the data files. When using image copies, RMAN copies byte for byte the live data files across the network to the auxiliary database.

In Oracle 12c the default format for transferring the data files is the backup set format (more on this shortly). In Oracle 11g, RMAN can only use the image copy format to transfer the data files. This process is depicted in Figure 4-1.

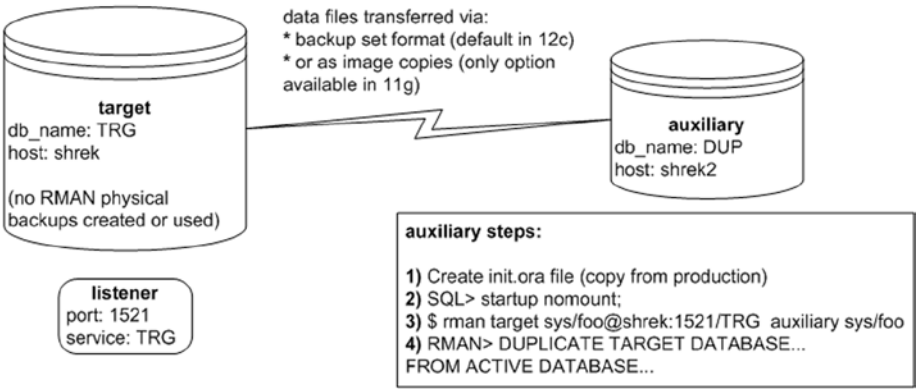

Figure 4-1. *RMAN active duplication*

Active database duplication using the backup set format is a feature of RMAN in Oracle 12c and higher. Using RMAN backup sets is usually the preferable format for performing active duplication. The backup set method doesn't copy a byte for byte copy of the data file, rather it copies the minimal amount of information in the data file that would be required to recreate the data file on the auxiliary database. This feature is referred to as *unused block compression*. This means if you have a large data file that is mostly empty, RMAN won't transfer the blocks that have never been used; only the blocks that have been used in the data file will be copied. This can greatly speed up the time and reduce the resources required to actively duplicate a database.

Additionally, the backup set format provides access to standard RMAN features such as compression, encryption, and parallel multisection transfer of large data files. For example, if your network bandwidth is constrained, you may want to consider having RMAN compress the backup set contents before copying them to the auxiliary database. This will use more CPU on the target and auxiliary databases as the compression and decompression take place, but the load on the network will be greatly reduced.

Oracle Net Configuration

Active duplication requires an RMAN connection to both the target and the auxiliary database. If you don't have network connectivity between the target and the auxiliary then active duplication is not possible. For instance, sometimes due to security reasons there is no network connectivity allowed between production and test environments. In those scenarios you'll have to use another approach like targetless duplication based on an RMAN backup (see Chapter 3 for details).

With active duplication the RMAN connection can be initiated from either the target or the auxiliary server. Recall the following points highlighted in Chapter 3 regarding where RMAN consumes the most resources when performing active duplication:

- When performing active duplication with the backup set format (default format in Oracle 12c and higher), most of the work is done by the auxiliary database channels.
- When performing active duplication when image copies are specified (only option available in Oracle 11g), the work is done by the target database channels.

Since a connection is required to both the target and the auxiliary, this means you have to connect to one of the databases over an Oracle Net connection. Oracle Net connectivity requires:

- A listener running on the database that you're connecting to over Oracle Net.
- A password file; any time you connect over Oracle Net to a database as a *SYSDBA* privileged user (such as *SYS*) or as a user with *SYSBACKUP* privileges you must have a password file in place. Active duplication requires that a password file exist for both the target and auxiliary servers.

In other words, whenever you run the RMAN client and need to connect over Oracle Net to either the target or the auxiliary database as a *SYS* (or *SYSBACKUP*) privileged user you must have Oracle Net set up and password files created.

Setting Up the Listener

This section describes the basics of setting up the listener. If you're not familiar with Oracle Net concepts then then refer to Chapter 6, which covers basic Oracle Net components and configuration.

For simple test environments you can start a default listener via:

```
$ lsnrctl start
```

If no *listener.ora* file has been configured for the listener, you'll initially see a message similar to this in the output:

```
The listener supports no services
```

If you haven't configured your listener to listen on a specific port then by default the listener will listen for incoming connections on port 1521. If listening on port 1521, by default the *LREG* process will register an actively running instance service with the listener (*PMON* will register an instance with the listener in Oracle 11g and before). The *LREG* process periodically (every 60 seconds) checks to see if any new instances are running on the server, and if so it will attempt to register an instance service with the listener. You can verify the registration of the service with the listener via:

```
$ lsnrctl services
```

After 60 seconds you should see each Oracle instance service registered with the listener in the output; for example:

```
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=shrek)(PORT=1521)))
Service "TRG" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

If you have configured the listener to listen on a port other than 1521, then you can instruct *LREG* (or *PMON* in Oracle 11g and before) to dynamically register a service with a listener by setting the *LOCAL_LISTENER* initialization parameter. For example, if you've configured the listener to listen on port 1522, then you can enable dynamic registration by setting the *LOCAL_LISTENER* initialization parameter as follows:

```
SQL> alter system set
local_listener='(ADDRESS=(PROTOCOL=TCP)(HOST=shrek)(PORT=1522))';
```

If your listener is listening on port 1521, ensure that the *LOCAL_LISTENER* parameter isn't set:

```
SQL> alter system set local_listener='';
```

Setting Up the Password File

In addition to Oracle Net being implemented, active duplication requires password files for both the target and auxiliary databases. To create a password file, do as follows:

```
$ cd $ORACLE_HOME/dbs
$ orapwd file=<filename> password=<password>
```

For example, for my database named *TRG* I created the password file as follows:

```
$ cd $ORACLE_HOME/dbs
$ orapwd file=orapwTRG password=foo
```

After the listener is started and a password file is in place, you can use the easy connect naming method to connect over Oracle Net to a remote database as a *SYSDBA* privileged user (such as *SYS*). For example, assuming the target database service name is *TRG*, host name is *shrek*, and port is 1521, here's what the connection looks like when initiating the RMAN client from the auxiliary server over Oracle Net to the target database:

```
$ rman target sys/foo@shrek:1521/TRG  auxiliary sys/foo
```

If initiating the connection from the target database, and assuming the auxiliary database service name is *DUP*, host name is *shrek2*, and port is 1521, the easy connection looks like this:

```
$ rman target sys/foo  auxiliary sys/foo@shrek2:1521/DUP
```

If you were initiating the RMAN connection from a client that doesn't reside on either the target server or the auxiliary server, you would then have to specify the Oracle Net connection information for both the target and the auxiliary. Also, a listener would need to be running on both the target server and the auxiliary server. Here's an example RMAN connection string for my environment:

```
$ rman target sys/foo@shrek:1521/TRG  auxiliary sys/foo@shrek2:1521/DUP
```

Another common method for specifying Oracle Net information is to use a *tnansames.ora* file on the client. This file is usually located in the *ORACLE_HOME/network/admin* directory and contains an alias for each service that is associated with network connection details, such as the host name, port number, and service name. For example:

```
TRG =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1521))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = TRG)))

DUP =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shrek2)(PORT = 1521))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = DUP)))
```

Once specified in the *tnsnames.ora* file, then the RMAN connection looks like this:

```
$ rman target sys/foo@TRG  auxiliary sys/foo@DUP
```

Again, if you need a primer on Oracle Net concepts then refer to Chapter 6. For full details on how to configure Oracle Net, refer to Oracle's documentation, which is freely available on the Oracle Technology Network website. My intention in this chapter is to provide you with just enough information to get started with active duplication.

Same Directory Structure and Database Name

This scenario duplicates the auxiliary database using the active (open) target database's data files and control files. Here are some points to be aware of with active duplication:

- No RMAN backup is required.
- If the target database is open then it is required to be in archivelog mode. When actively duplicating an open database, if it's not in archivelog mode, you'll receive this message:

  ```
  ORA-19602: cannot backup or copy active file in NOARCHIVELOG mode
  ```

- Does require Oracle Net connectivity between the target and auxiliary.
- Does require a password file for target and auxiliary database. If there's no password file in place, you'll receive this message:

  ```
  ORA-17627: ORA-01017: invalid username/password; logon denied
  ```

The required steps can be performed from either the target (source) server or the auxiliary (destination) server. Figure 4-2 depicts the scenario in which all of the steps are performed on the auxiliary server. This example uses an Oracle 12c database, therefore the default format for the data file transfer is the backup set format. If this were an Oracle 11g database the image copy format would be used for the data file transfer. In this example the directory structure and database name remain the same from target to auxiliary.

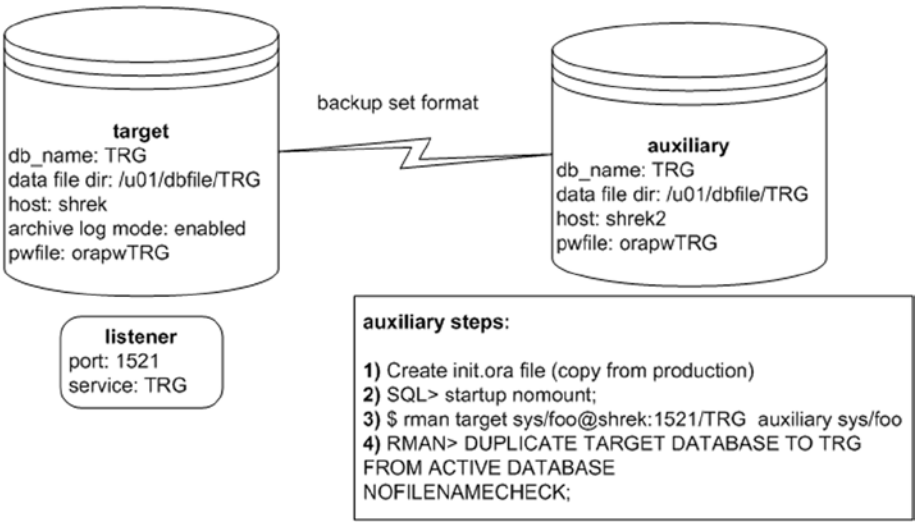

Figure 4-2. *Active duplication with same database name and directory structure*

The steps for duplication for this scenario are described in detail next.

1. Copy the initialization file from the target to the auxiliary. You can use either an SPFILE or a text-based *init.ora* file. Here's an example of using the Linux/UNIX *scp* command for my environment. This command is initiated from the auxiliary server:

   ```
   $ cd $ORACLE_HOME/dbs
   $ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora.
   ```

2. Start up the auxiliary database in nomount mode:

   ```
   $ sqlplus / as sysdba
   SQL> startup nomount;
   SQL> exit;
   ```

3. When connecting to RMAN you can initiate the connection from the target, the auxiliary, or a remote client. This shows an example of connecting to target and auxiliary initiated from the auxiliary server:

   ```
   $ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo
   ```

If you were issuing the RMAN connection from the target, it would look like this (assuming there's a listener on the auxiliary host listening on port 1521):

```
$ rman target sys/foo auxiliary sys/foo@shrek2:1521/TRG
```

For active duplication you must specify a username and password for both the target and the auxiliary connection. Otherwise, when issuing the *DUPLICATE* command RMAN will throw an *RMAN-05609* or an *RMAN-05610* error specifying that the username and password are required.

4. Now execute the *DUPLICATE* command:

```
RMAN> DUPLICATE TARGET DATABASE TO TRG
FROM ACTIVE DATABASE
NOFILENAMECHECK;
```

You should now see a great deal of output. Here's a small snippet:

```
Starting Duplicate Db at ...
using target database control file instead of recovery catalog
...
channel ORA_AUX_DISK_1: using network backup set from service shrek:1521/TRG
...
Cannot remove created server parameter file
Finished Duplicate Db at ...
```

After it successfully completes, on the auxiliary server you should have an identical copy of the database from the target database. Don't worry about the *Cannot remove create server parameter file* message. RMAN is just telling you that it created an SPFILE as part of the duplication process and did not remove it (which is fine).

Different Directory Structure and Database Name Using SPFILE Clause

In this example active duplication is performed where the directory structures and database names are different from the target to the auxiliary. This type of operation is best achieved by using the *SPFILE* clause of the *DUPLCIATION* command. This approach requires the target (source) database be started using an SPFILE; if you don't start the target database with an SPFILE, you'll receive the following error when duplicating:

```
RMAN-05557: Target instance not started with server parameter file
```

This approach also requires that the auxiliary (destination) database *not be* started with an SPFILE; if you start your auxiliary instance with an SPFILE, you'll receive this error when duplicating:

```
RMAN-05537: DUPLICATE without TARGET connection when auxiliary
instance is started with spfile cannot use SPFILE clause
```

This error message states that you can't duplicate without a target connection. In this scenario you will be connected to the target, so don't let that confuse you. The point here is that you must start the auxiliary with a text-based *init.ora* file and not an SPFILE.

The required steps can be performed from either the target server or the auxiliary server. Figure 4-3 graphically displays this method.

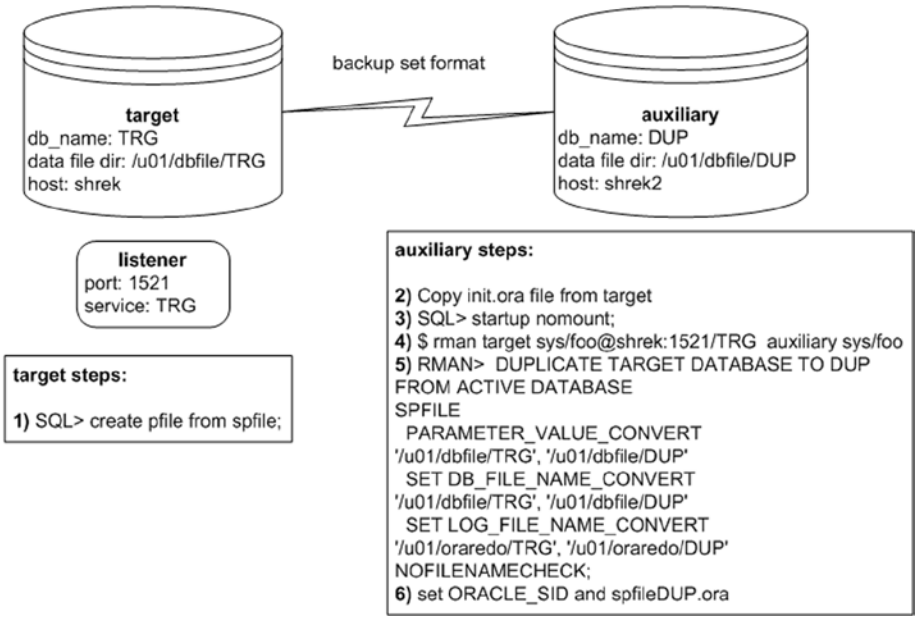

Figure 4-3. *Active duplication with different database name and directory structure*

The steps required for this scenario are described in detail next. This example assumes the target database is using an SPFILE. You can verify this on the target as follows:

```
SQL> show parameter spfile;
NAME        TYPE        VALUE
---------   ----------- ------------------------
spfile      string      /orahome/app/oracle/product
                        /12.1.0.2/db_1/dbs/spfileTRG.ora
```

1. On the target database, create a text-based initialization file:

   ```
   SQL> create pfile from spfile;
   ```

2. Copy the initialization file from the target to the auxiliary. Here's an example of using the Linux/UNIX *scp* command for my environment initiated from the auxiliary server:

```
$ cd $ORACLE_HOME/dbs
$ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora.
```

3. Start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

4. Connect to RMAN and to target and auxiliary databases. This shows the connection originating from the auxiliary (destination) server:

```
$ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo
```

If you were issuing the RMAN connection from the target, it would look like this:

```
$ rman target sys/foo auxiliary sys/foo@shrek2:1521/TRG
```

5. Now execute the *DUPLICATE* command. This example assumes that the target database is using an SPFILE, and therefore the *SPFILE* clause of the *DUPLICATE* command can be used to specify the new directory path names for the control files (*PARAMETER_VALUE_CONVERT*), data files (*DB_FILE_NAME_CONVERT*), and online redo log files (*LOG_FILE_NAME_CONVERT*):

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
FROM ACTIVE DATABASE
SPFILE
  PARAMETER_VALUE_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET DB_FILE_NAME_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET LOG_FILE_NAME_CONVERT
'/u01/oraredo/TRG', '/u01/oraredo/DUP'
NOFILENAMECHECK;
```

You'll see a great volume of output. Here is a small portion for this example:

```
Starting Duplicate Db at ...
using target database control file instead of recovery catalog
allocated channel: ORA_AUX_DISK_1
...
```

```
executing Memory Script
database opened
Finished Duplicate Db at ...
```

After RMAN successfully completes the database duplication, on the auxiliary server you should have a complete copy of the target with a different name and files located in a different directory structure.

Note that *NOFILENAMECHECK* was also used. Why is that required when the directory structures are different from the target to the auxiliary? You need the *NOFILENAMECHECK* any time you experience the *RMAN-05001* error. That error may be thrown in this scenario because RMAN detects that the archive redo log location is the same on the target as on the auxiliary. This is not an issue, you just need to include *NOFILENAMECHECK* to prevent RMAN from throwing the error.

Also keep in mind that if you have multiple directories that you need to transform, you need to add them where appropriate; for example:

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
FROM ACTIVE DATABASE
SPFILE
  PARAMETER_VALUE_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP',
'/u02/dbfile/TRG', '/u02/dbfile/DUP'
  SET DB_FILE_NAME_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP',
'/u02/dbfile/TRG', '/u02/dbfile/DUP'
  SET LOG_FILE_NAME_CONVERT
'/u01/oraredo/TRG', '/u01/oraredo/DUP',
'/u02/oraredo/TRG', '/u02/oraredo/DUP'
NOFILENAMECHECK;
```

6. To start your database with the new name of *DUP*, exit from RMAN, then get into SQL*Plus and shut down the database, rename the SPFILE, and export your *ORACLE_SID* to reflect the new instance name and restart your database:

```
RMAN> exit;
$ sqlplus / as sysdba
SQL> shutdown immediate;
SQL> exit;
$ cd $ORACLE_HOME/dbs
$ mv spfileTRG.ora spfileDUP.ora
$ export ORACLE_SID=DUP
$ sqlplus / as sysdba
SQL> startup
```

In the prior code, you may be asking, "Why not issue the *SHUTDOWN* command from within RMAN?" If you issue a *SHUTDOWN* while connected to the target database, RMAN will stop the target database (and not the auxiliary database), which is probably not the behavior that you want.

You might also wonder if prior to issuing the *DUPLICATE* command is it possible to first set the *ORACLE_SID* to the duplicated database name (*DUP* in this example), and use an initialization file named *initDUP.ora* to start the auxiliary in nomount mode? The answer is yes, in that scenario you need to ensure a password file has been created with the name of *orapwDUP* before duplicating.

■ **Note** With active duplication RMAN doesn't allow you to restore to a point in time in the past. If you attempt a point–in-time duplication operation, RMAN will throw an *RMAN-05600* error indicating you cannot specify *UNTIL* clause when duplicating from active database.

Replicating from a Noarchivelog Mode Target

My manager pings me every once in a while and asks me to replicate a database from one server to another. The target database is usually some sort of a test database and is not in archivelog mode. One might wonder if it's possible to use the RMAN duplication functionality in this scenario. To test this, first connect to the target and the auxiliary:

```
$ rman target sys/foo@TRG auxiliary sys/foo
```

Next, attempt to run the *DUPLICATE* command:

```
RMAN> DUPLICATE TARGET DATABASE TO DUP...
```

If the target (active) database is in noarchivelog mode and open, the RMAN throws this error:

```
ORA-19602: cannot backup or copy active file in NOARCHIVELOG mode
```

One solution to this situation is to place the target database in mount mode while the duplication is taking place. Figure 4-4 depicts the steps involved in this situation.

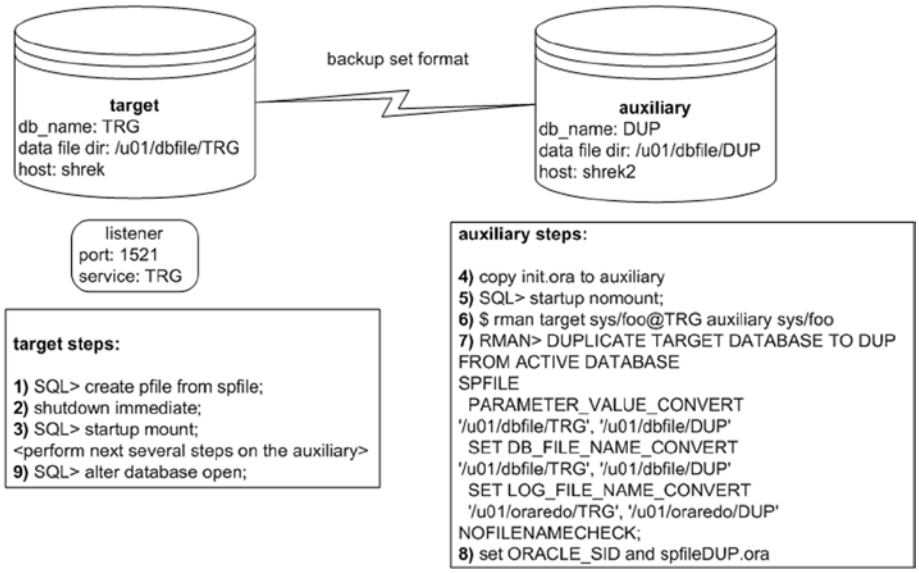

Figure 4-4. *Duplicating an active noarchivelog mode database*

The steps required for this scenario are described in detail next. This example assumes the target database is using an SPFILE. You can verify this on the target as follows:

```
SQL> show parameter spfile;
NAME        TYPE        VALUE
---------   ----------- ------------------------
spfile      string      /orahome/app/oracle/product
                        /12.1.0.2/db_1/dbs/spfileTRG.ora
```

1. On the target database, create a text-based initialization file:

   ```
   SQL> create pfile from spfile;
   ```

2. On the target database, shut it down with the *IMMEDIATE* option:

   ```
   SQL> shutdown immediate;
   ```

3. Now start the target database up in mount mode:

   ```
   SQL> startup mount;
   SQL> exit;
   ```

4. Copy the initialization file from the target to the auxiliary. Here's an example of using the Linux/UNIX *scp* command for my environment. This command is initiated from the auxiliary server:

```
$ cd $ORACLE_HOME/dbs
$ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora.
```

You must use a text-based initialization file to start the auxiliary database in this scenario. If you use an SPFILE to start the auxiliary database you'll receive this error when duplicating:

```
RMAN-05537: DUPLICATE without TARGET connection when auxiliary instance
is started with spfile cannot use SPFILE clause
```

5. Now start the auxiliary in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

6. Next, connect to target and auxiliary databases. This shows the connection originating from the auxiliary database server:

```
$ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo
```

7. Now execute the RMAN *DUPLICATE* command. This example assumes that the target database is using an *SPFILE*, and therefore the *SPFILE* clause of the *DUPLICATE* command can be used to specify the new directory path names for the control files (*PARAMETER_VALUE_CONVERT*), data files (*DB_FILE_NAME_CONVERT*), and online redo log files (*LOG_FILE_NAME_CONVERT*):

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
FROM ACTIVE DATABASE
SPFILE
  PARAMETER_VALUE_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET DB_FILE_NAME_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET LOG_FILE_NAME_CONVERT
  '/u01/oraredo/TRG', '/u01/oraredo/DUP'
NOFILENAMECHECK;
```

At this point you'll see a great deal of messaging. Here is a small bit of the output:

```
Starting Duplicate Db at ...
using target database control file instead of recovery catalog
...
executing Memory Script
database opened
Finished Duplicate Db at ...
```

After it successfully completes, on the auxiliary server you should have a copy of the target with a different name and in a different directory structure.

8. To start your database with the new instance name of *DUP*, exit from RMAN, then get into SQL*Plus and shut down the database, rename the SPFILE, and export the *ORACLE_SID* operating system variable to reflect the new instance name and restart your database:

```
$ sqlplus / as sysdba
SQL> shutdown immediate;
SQL> exit;
$ cd $ORACLE_HOME/dbs
$ mv spfileTRG.ora spfileDUP.ora
$ export ORACLE_SID=DUP
$ sqlplus / as sysdba
SQL> startup
```

9. As the last step, don't forget to open the target database for use again. On the target server, issue the following SQL command:

```
SQL> alter database open;
```

Same-Host Replication

You may have the requirement to replicate a database where the target and the auxiliary are on the same server. Figure 4-5 depicts this concept.

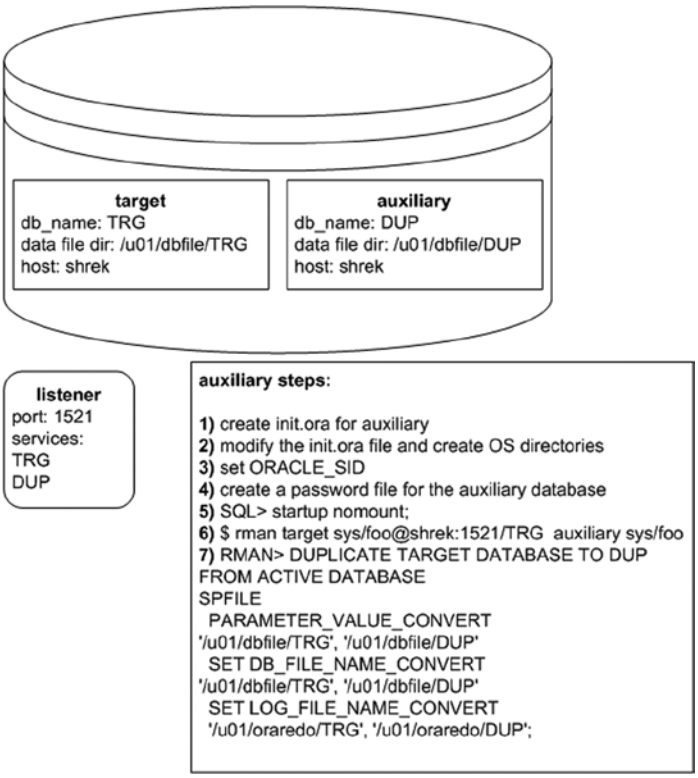

Figure 4-5. *Same-host active database duplication*

The steps required for this scenario are described in detail next. This example assumes the target database is using an SPFILE. You can verify this on the target as follows:

```
SQL> show parameter spfile;
NAME        TYPE        VALUE
---------   ----------- ------------------------
spfile      string      /orahome/app/oracle/product
                        /12.1.0.2/db_1/dbs/spfileTRG.ora
```

1. First create an *init.ora* file for the auxiliary database. The easiest way to create a text-based parameter file from the SPFILE and then rename it:

```
$ sqlplus / as sysdba
SQL> create pfile from spfile;
SQL> exit;
$ cd $ORACLE_HOME/dbs
$ cp initTRG.ora initDUP.ora
```

2. Edit the *initDUP.ora* with an operating system utility (such as *vi* or *notepad*) and modify the *DB_NAME* parameter and any parameters that contain directory structures (like *CONTROL_FILES*, *LOG_ARCHIVE_DEST_1*, and so on). Here are what the lines look like for this example:

```
db_name='DUP'
control_files='/u01/dbfile/DUP/control01.ctl','/u01/dbfile/DUP/
control02.ctl'
log_archive_dest_1='LOCATION=/u01/arch/DUP'
```

Ensure that the directories being referenced are not the target database directories. If this is the first time you're performing the duplication on the server then you'll have to create the new directories with an operating system utility (like *mkdir*):

```
$ mkdir /u01/dbfile/DUP
$ mkdir /u01/oraredo/DUP
$ mkdir /u01/arch/DUP
```

3. Set your *ORACLE_SID* variable to point to the auxiliary instance:

```
$ export ORACLE_SID=DUP
```

4. Create a password file for the auxiliary database:

```
$ cd $ORACLE_HOME/dbs
$ orapwd file=orapwDUP password=foo
```

5. Start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

If you get an error like the following snippet, then ensure that you don't have any parameters defined incorrectly in the auxiliary *init.ora* file:

```
ORA-01078: failure in processing system parameters
```

6. Connect to the target and the auxiliary instances:

```
$ rman target sys/foo@shrek:1521/TRG  auxiliary sys/foo
```

7. Issue the *DUPLICATE* command:

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
FROM ACTIVE DATABASE
SPFILE
  PARAMETER_VALUE_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET DB_FILE_NAME_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET LOG_FILE_NAME_CONVERT
  '/u01/oraredo/TRG', '/u01/oraredo/DUP';
```

At this point you'll see a large amount of output; here's a tiny snip:

```
Starting Duplicate Db at ...
using target database control file instead of recovery catalog
...
database opened
Finished Duplicate Db at ...
```

If you receive an error like the following:

```
RMAN-05520: database name mismatch, auxiliary instance has TRG,
command specified DUP
```

In this situation ensure that your *ORACLE_SID* is set to *DUP* and that the *DB_NAME* parameter in the *init.ora* file is set to *DUP*:

You may also receive a memory error such as the following:

```
ORA-27125: unable to create shared memory segment
```

This indicates you don't have enough physical memory on the host to start two instances. You'll need to start any existing instances with less memory or add physical memory to the server.

You must also ensure that the auxiliary database is started with a text-based *init.ora* file, otherwise you'll receive this error:

```
RMAN-05537: DUPLICATE without TARGET connection when auxiliary instance
is started with spfile cannot use SPFILE clause
```

You may receive an *RMAN-05001* error like the following:

```
RMAN-05001: auxiliary file name ... conflicts with a file
used by the target database
```

If this happens ensure that in the *DUPLICATE* command you've correctly mapped the target directories to new auxiliary directories.

If you continue to receive the *RMAN-05001* error, and are sure that you've correctly mapped the target directories to the new auxiliary directories then consider adding the *NOFILENAMECHECK* clause. Keep in mind this can be dangerous when performing same server replication as it instructs RMAN not to check if the file exists before writing it to disk:

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
FROM ACTIVE DATABASE
SPFILE
  PARAMETER_VALUE_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET DB_FILE_NAME_CONVERT
'/u01/dbfile/TRG', '/u01/dbfile/DUP'
  SET LOG_FILE_NAME_CONVERT
  '/u01/oraredo/TRG', '/u01/oraredo/DUP'
NOFILENAMECHECK;
```

After the duplication process has finished, check the *alert.log* and ensure there have been no errors. You may see an error such as this:

```
ORA-01110: data file 201: '/u01/dbfile/DUP/temp01.dbf'
```

To resolve prior error either move or remove the existing temporary tablespace temp file using operating system commands, for example:

```
$ mv /u01/dbfile/DUP/temp01.dbf /u01/dbfile/DUP/tempo01.old.dbf
```

Then stop and start the database. When starting Oracle will detect the temporary tablespace temp file is missing and recreate it.When complete, you should have a fully functioning copy of the target database named *DUP* that resides on the target server.

Image Copies

In Oracle 12c and above, when using active database duplication RMAN will by default use the backup set format to copy data files from the target database to the source database. If you want to ensure that image copies be used, then you need to explicitly instruct RMAN to do this with the *USING COPY* clause. Figure 4-6 illustrates the steps involved with this type of duplication.

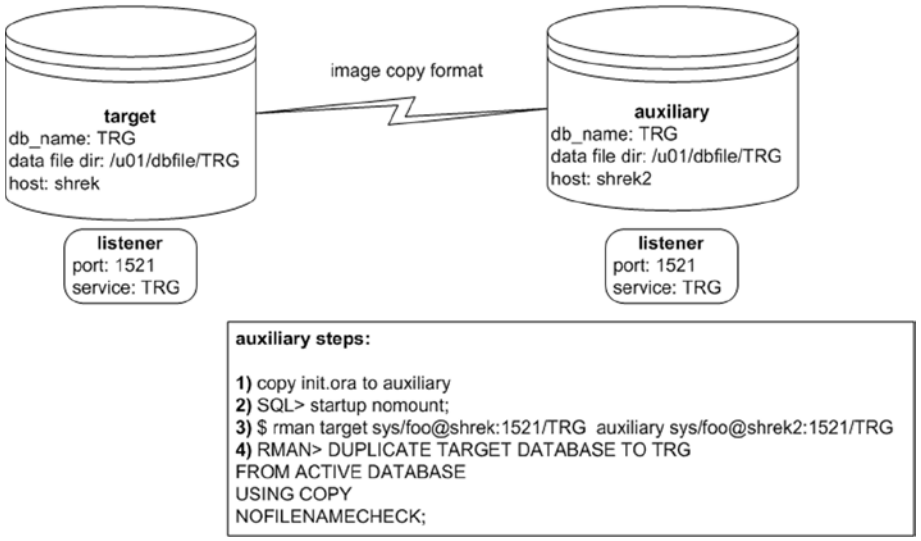

Figure 4-6. *Active database duplication with image copies*

1. Copy the initialization file from the target to the auxiliary. You can use either an SPFILE or a text-based *init.ora file*. Here's an example of using the Linux/UNIX *scp* command for my environment. This command is initiated from the auxiliary server:

   ```
   $ cd $ORACLE_HOME/dbs
   $ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora.
   ```

2. Start up the auxiliary database in nomount mode:

   ```
   $ sqlplus / as sysdba
   SQL> startup nomount;
   SQL> exit;
   ```

3. Connect to both the target and the auxiliary using an Oracle Net connection; for example:

   ```
   $ rman target sys/foo@shrek:1521/TRG  auxiliary
   sys/foo@shrek2:1521/TRG
   ```

In this scenario, you must specify an Oracle Net connection for both the target database and the auxiliary database, otherwise you'll receive this error when you attempt to run the *DUPLICATE* command:

```
RMAN-06217: not connected to auxiliary database with a net service name
```

You may also receive this error when connecting to the auxiliary instance:

```
RMAN-04006: error from auxiliary database:
ORA-12528: TNS:listener: all appropriate instances are
blocking new connections
```

In this situation statically register the auxiliary instance service with the listener. For details on static service registration see Chapter 6.

4. Now issue the *DUPLICATE* command with the *USING COPY* clause:

```
RMAN> DUPLICATE TARGET DATABASE TO TRG
FROM ACTIVE DATABASE
USING COPY
NOFILENAMECHECK;
```

The output of this command verifies that image copies are being created; for instance:

```
channel ORA_DISK_1: starting datafile copy
input datafile file number=00001 name=/u01/dbfile/TRG/system01.dbf
```

Recall that an image copy is a byte-for-byte copy of the data file. Therefore, you may find that this method takes longer than it would have had you used the backup set format. I recommend that you use the default backup set format, as this is more efficient (typically uses fewer resources because RMAN is not copying the data files block for block). However, if you're using an older version of Oracle, then the image copy format is the only one available for active database duplication over the network.

■ **Note** In Oracle 11g, image copies are the only format available with active duplication.

Compression

When using active duplication with the backup set format, you have the option of using compression. If you have limited bandwidth on the network connection between the target and the auxiliary, you may want to consider using compression. The compression option will transfer the data files across the network in a compressed backup set format. This will minimize the impact on the network. Figure 4-7 illustrates this concept.

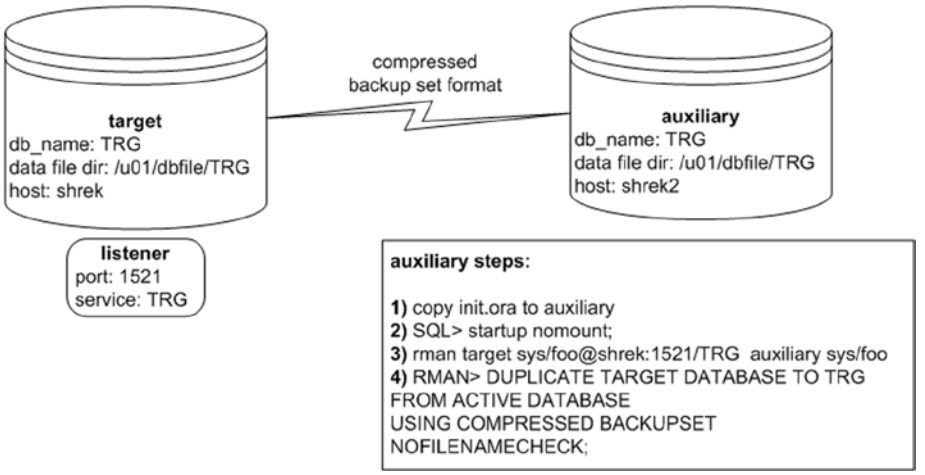

Figure 4-7. *Active duplication using compressed backup set format*

Keep in mind that the backup set format for transferring data files is only available in Oracle 12c and higher. The detailed steps for the scenario are described next.

1. First copy the initialization file from the target to the auxiliary. You can use either an SPFILE or a text-based *init.ora* file. Here's an example of using the Linux/UNIX *scp* command for my environment. This command is initiated from the auxiliary server:

   ```
   $ cd $ORACLE_HOME/dbs
   $ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora.
   ```

2. Start up the auxiliary in nomount mode:

   ```
   $ sqlplus / as sysdba
   SQL> startup nomount;
   SQL> exit;
   ```

3. Next, connect to the target and the auxiliary:

   ```
   $ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo
   ```

4. Then issue the *DUPLICATE* command, specifying compression:

   ```
   RMAN> DUPLICATE TARGET DATABASE TO TRG
   FROM ACTIVE DATABASE
   USING COMPRESSED BACKUPSET
   NOFILENAMECHECK;
   ```

In the lengthy output, you should see that the compressed backup set format is used to transfer the data:

```
channel ORA_AUX_DISK_1: using compressed network backup set from service trg
```

There are several levels of compression available. When you specify *USING COMPRESSED BACKUPSET*, if no compression has been configured, then by default RMAN uses the *BASIC* compression. You can verify the compression algorithm in use via:

```
RMAN> show compression algorithm;
```

Here is some output verifying that the *BASIC* compression algorithm is in use:

```
RMAN configuration parameters for database with db_unique_name TRG are:
CONFIGURE COMPRESSION ALGORITHM 'BASIC' AS OF RELEASE 'DEFAULT' OPTIMIZE
FOR LOAD TRUE ; # default
```

The other compression options are *LOW*, *MEDIUM*, and *HIGH*. Here's an example of configuring RMAN to use the *MEDIUM* compression algorithm:

```
RMAN> CONFIGURE COMPRESSION ALGORITHM 'MEDIUM';
```

You can set the compression algorithm back to the default (of *BASIC*) via:

```
RMAN> configure compression algorithm clear;
```

In summary, compression is a good choice when you know your network bandwidth is the bottleneck. Compression will reduce the number of bytes transferred from the target to the auxiliary. The downside is that compression requires more CPU resources on the target server to compress the backup set data and then more CPU again to uncompress the data on the auxiliary server.

■ **Note** The *BASIC* encryption is freely available. If you want to use the *LOW*, *MEDIUM*, and *HIGH* compression options you need a license for the Oracle Advanced Compression option. The Advanced Compression option is only available with the Enterprise Edition of Oracle.

Encryption

Starting with Oracle 12c, you have the option of using encryption when using active duplication with the backup set format. This provides additional security when duplicating a database from the target server to the auxiliary server.

When using transparent encryption, you must copy the keystore that contains the encryption key to the auxiliary server (placed in the default location or in the location specified in the *sqlnet.ora* file). The keystore must also be an automatic login keystore.

Once the keystore has been copied from the target server to the auxiliary server, on the auxiliary connect to the target and the auxiliary databases:

```
$ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo
```

Next, issue the *DUPLICATE* command:

```
RMAN> DUPLICATE TARGET DATABASE TO TRG
FROM ACTIVE DATABASE
NOFILENAMECHECK;
```

This command uses the backup set format to transfer the data files from the target to the auxiliary in an encrypted manner.

■ **Tip** See the *Oracle Database Advanced Security Guide* for details on using encryption. The Advanced Security option requires an additional license from Oracle. The Advanced Security option is only available with the Enterprise Edition of Oracle.

Summary

This chapter has focused on active database duplication. In this mode there's no requirement to have an RMAN backup available. Rather, RMAN uses the live (active target) database data files and control files for the source. RMAN can transfer the data files across the network in either an RMAN backup set format or in an image copy format. Using the RMAN backup set format has many advantages over image copies:

- It allows you to use advanced features such as compression, encryption, and parallel multisection transfer of large data files.
- It only copies blocks that have ever been used within the data file. Depending on the used space in the data file, this can be more efficient than full image copies.

There were many active duplication examples shown in this chapter. The simplest example is where the database name and directory structure are identical on both the target and the auxiliary servers. More advanced examples detailed how to use active database replication in environments where it's required to have a different auxiliary database name (from the target) and where the auxiliary server has a different directory structure. You should be able to modify these examples to fit the requirements of your environment.

CHAPTER 5

Advanced Topics

Chapters 3 and 4 discussed duplicating a database from RMAN backups and from active databases, respectively. This chapter focuses on advanced topics such as:

- Partially duplicating a subset of tablespaces
- Duplicating in parallel
- Creating Data Guard standby databases with the *DUPLICATE* command
- Duplicating container and pluggable databases
- Duplicating databases between RAC and ASM to non-RAC and non-ASM (and vice versa)

The RMAN *DUPLICATE* command helps automate what would otherwise be quite complex tasks. First up is the topic of partial database duplication.

Partial Database Duplication

There are a couple of different techniques for duplicating a subset of tablespaces from the target (source) to the auxiliary (destination). You can either exclude certain tablespaces or just include specific tablespaces. Examples of each technique are shown in the following sections.

Excluding Tablespaces

When excluding tablespaces from a duplication operation, you can specify tablespaces to be skipped or skip read-only tablespaces. Examples of each will be shown in this section. Skipping specific tablespaces is enabled via the *SKIP TABLESPACE* clause. The following example duplicates while skipping the *USERS* tablespace. The steps required to do this are detailed next:

1. Since this example uses targetless duplication it assumes there is a backup of the target database available to the auxiliary database (see Chapter 3 for complete details regarding targetless duplication). On the target database, create backups and copy them to the auxiliary server (or to network storage that is readable by the auxiliary server):

```
$ rman target/
RMAN> backup database plus archivelog;
RMAN> exit;
```

You can verify the location of the backups via the *LIST BACKUP* command; here's a small snippet of the output:

```
Piece Name: /u01/rman/TRG/TRGrman1_rrpth9n6_1_1.bk
```

In my environment, the RMAN backups on the target host are in the */u01/rman/TRG* directory. Next, I copy the backup files from the target to the auxiliary using the Linux/UNIX *scp* command. You need to ensure the destination directory exists before initiating the *scp* command. This command is initiated from the auxiliary server:

```
$ scp oracle@shrek:/u01/rman/TRG/*.*  /u01/rman/TRG
```

2. Now copy the initialization file from the target to the auxiliary. You can use either an SPFILE or a text-based *init.ora* file. Here's an example of using the Linux/UNIX *scp* command for my environment. This command is initiated from the auxiliary server:

```
$ cd $ORACLE_HOME/dbs
$ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora.
```

3. Start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

4. Next, connect to the auxiliary database via RMAN:

```
$ rman auxiliary /
```

5. This example uses an RMAN backup for the source of the control file and data files, and therefore no connection to the target database is necessary. Once connected to the auxiliary, issue the *DUPLICATE* command, as follows:

```
RMAN> duplicate database to TRG
backup location '/u01/rman/TRG'
skip tablespace users
nofilenamecheck;
```

Once complete, a quick SQL query verifies that the *USERS* tablespace was not duplicated:

```
RMAN> select tablespace_name from dba_tablespaces
      where tablespace_name='USERS';

no rows selected
```

■ **Tip** See MOS note 1355120.1 for further details regarding skipping tablespaces during duplication.

It's also possible to skip read-only tablespaces during the duplication process. The following command performs active duplication while skipping any tablespaces that have been placed in read-only mode:

```
RMAN> DUPLICATE TARGET DATABASE TO TRG
FROM ACTIVE DATABASE
SKIP READONLY
NOFILENAMECHECK;
```

When you exclude read-only tablespaces during the duplication command, you can expect to see output that indicates that the read-only tablespaces have been skipped:

```
datafile 5 not processed because file is read-only
datafile 6 not processed because file is read-only
```

To wrap up this section, excluding tablespaces is a good method for performing partial database duplication when you have specific tablespaces that you know aren't required in the copied database.

Including Tablespaces

Another way to skip tablespaces is to instruct RMAN which tablespaces to include in the *DUPLICATE* command. In this manner, any tablespaces not included will be excluded. This example instructs RMAN to include the *REPDATA* and *REPIDX* tablespaces. The steps required to do this are detailed next:

1. Copy the initialization file from the target to the auxiliary. You can use either an SPFILE or a text-based *init.ora* file. Here's an example of using the Linux/UNIX *scp* command for my environment. This command is initiated from the auxiliary server:

   ```
   $ cd $ORACLE_HOME/dbs
   $ scp oracle@shrek:$ORACLE_HOME/dbs/initTRG.ora.
   ```

2. Start up the auxiliary database in nomount mode:

   ```
   $ sqlplus / as sysdba
   SQL> startup nomount;
   SQL> exit;
   ```

3. Next, connect to the auxiliary database via RMAN:

   ```
   $ rman auxiliary /
   ```

4. This example uses an RMAN backup for the source of the control file and data files, and therefore no connection to the target database is necessary. Thus, you need to ensure a valid backup of the target database is available to the auxiliary database (see Chapter 3 for complete details on targetless duplication). Once connected to the auxiliary, issue the *DUPLICATE* command, as follows:

   ```
   RMAN> duplicate database to TRG
   backup location '/u01/rman/TRG'
   tablespace repdata, repidx
   nofilenamecheck;
   ```

In this situation, RMAN duplicates all essential tablespaces (*SYSTEM*, *SYSAUX*, *UNDO*, and *TEMP*) and then only the tablespaces included in the *TABLESPACE* clause. A quick SQL query verifies this:

```
RMAN> select tablespace_name from dba_tablespaces;

TABLESPACE_NAME
------------------------------
SYSTEM
SYSAUX
UNDOTBS1
TEMP
REPDATA
REPIDX
```

Including specific tablespaces provides a quick method for creating a copy of a database that only contains the tablespaces required in the duplicated environment.

Parallelism

Parallelism can be used to greatly increase the performance of any RMAN backup, restore, or duplication operation. First I'll cover general parallelism techniques as they apply to duplication and then I'll review parallelism using the *SECTION SIZE* clause.

Configuring Parallelism

If you're performing an active duplication (meaning you are connected to both the target and auxiliary and are using the live target database as the source for the duplication), then RMAN will automatically allocate the number of channels per the degree of parallelization configured in the target database. For example, if the degree of parallelism has been defined in the target database to be 4 (as follows):

```
RMAN> configure device type disk parallelism 4;
```

then when performing active replication, both the target and the auxiliary database will have 4 channels opened.

If performing targetless duplication (meaning you're restoring from an RMAN backup), RMAN will open the number of channels configured for RMAN that were defined in the target at the time the backup was taken.

You can manually specify the channels to be opened via *ALLOCATE* within a run block. For example, first connect to the auxiliary:

```
$ rman auxiliary /
```

Then allocate channels and run the *DUPLICATE* command from within a run block:

```
RMAN> run{
allocate auxiliary channel dup1 type disk;
allocate auxiliary channel dup2 type disk;
allocate auxiliary channel dup3 type disk;
allocate auxiliary channel dup4 type disk;
#
duplicate database to TRG
backup location '/u01/rman/TRG'
nofilenamecheck;
}
```

You can verify the number of channels open. From another session, run this query:

```
SET LINES 132
COL username    FORM a10
COL kill_string FORM A12
COL os_id       FORM A6
COL client_info FORM A28
COL action      FORM A21
--
SELECT
 a.username
,a.sid || ',' || a.serial# AS kill_string
, b.spid AS OS_ID
,(CASE WHEN a.client_info IS NULL AND a.action IS NOT NULL
  THEN 'First Default'
       WHEN a.client_info IS NULL AND a.action IS NULL
  THEN 'Polling'
  ELSE a.client_info
  END) client_info
,a.action
FROM v$session a
    ,v$process b
WHERE a.program like '%rman%'
AND  a.paddr = b.addr;
```

Here is some sample output:

```
USERNAME   KILL_STRING  OS_ID  CLIENT_INFO               ACTION
---------- ------------ ------ ------------------------- -------------------
SYS        21,48425     12664  rman channel=dup1         0000023 STARTED40
SYS        23,60820     12665  rman channel=dup2         0000016 STARTED40
SYS        20,7160      12663  Polling
SYS        25,21933     12667  rman channel=dup4         0000008 FINISHED129
SYS        24,22053     12666  rman channel=dup3         0000008 FINISHED129
SYS        1,17710      12662  First Default             0000041 FINISHED64
```

Parallelism can greatly reduce the time required to duplicate a database. This is especially true for servers that have multiple CPUs and data files spread across multiple locations.

Using SECTION SIZE

By default RMAN uses at most one channel (process) to back up a data file. Starting with Oracle 11g, RMAN added the capability of using multiple processes to back up one data file. This is accomplished by specifying the *SECTION SIZE* clause. Using parallelism along with *SECTION SIZE* instructs RMAN to use multiple channels to back up a datafile. For example, say you had a 32-gig data file you wanted to back up. If you specify a *SECTION SIZE* of 8 gigs and a degree of parallelism of 4, then RMAN can use four parallel processes, each working on an 8-gig section of the data file. This feature is known as a *multisection backup*. The idea here is that you increase the performance of a backup by using multiple channels to back up one data file.

Starting with Oracle 12c, RMAN extended the multisection feature to active database duplication. This means you can specify the *SECTION SIZE* when replicating from an active database to parallelize the duplication of large data files. When you specify a *SECTION SIZE* and configure parallelism this instructs RMAN to use multiple channels to duplicate a single data file.

For example, say you have a database with 32 gigs of data files and you require RMAN to duplicate these data files in parallel. Further assume that the primary target database has never configured parallelism; therefore, you'll need to manually allocate channels on the primary database as well as on the auxiliary database. The following specifies a *SECTION SIZE* of 8 gig and parallelism of four channels:

```
RMAN> run{
allocate channel prmy1 type disk connect 'sys/"foo"@TRG';
allocate channel prmy2 type disk connect 'sys/"foo"@TRG';
allocate channel prmy3 type disk connect 'sys/"foo"@TRG';
allocate channel prmy4 type disk connect 'sys/"foo"@TRG';
allocate auxiliary channel dup1 type disk;
allocate auxiliary channel dup2 type disk;
allocate auxiliary channel dup3 type disk;
allocate auxiliary channel dup4 type disk;
DUPLICATE TARGET DATABASE TO TRG
FROM ACTIVE DATABASE
SECTION SIZE 8G
NOFILENAMECHECK;
}
```

In this way you can increase the performance when duplicating databases containing large data files.

Creating Standby Databases

Data Guard is Oracle's flagship disaster recovery tool. One main feature of this tool is the ability to implement a *standby database*. A standby database is a near real time copy of the primary (source) database. The standby database is typically created on a different server in a different data center from the primary database. The idea is that if a disaster struck (e.g., flood, earthquake, long-term power outage, and so on) you could quickly point applications at the standby database and be back in business literally within seconds or minutes, depending on how you've configured the Data Guard environment.

You can manually create a standby database or use the RMAN *DUPLICATE* command. A high-level description of how to manually implement a standby database is required to understand the utility of using *DUPLICATE*. Here's an overview of the manual steps:

1. Ensure Oracle Net connectivity exists between the primary server and standby server.
2. Create standby redo logs on the primary database (optional, but recommended).
3. Configure the initialization parameter files.
4. Create a physical backup of the primary database (hot, cold, or RMAN backup).
5. Create a standby control file.
6. Copy the standby control file and backup to the standby server.
7. Restore and recover the standby database.
8. Start the Data Guard processes that keep the primary and standby database synchronized.
9. Troubleshoot any issues.

These steps aren't super difficult, but they do require solid DBA backup and recovery skills. Manually creating a standby can be a time-consuming process, especially the steps where you're creating a backup of the primary and restoring and recovering that backup to the standby server. Enter the RMAN *DUPLICATE* command. In one command consisting of just a few lines of code, you can automate the creation of the standby database. You can create a standby by duplicating from an active target (primary) database, or you can create the standby from an RMAN backup of the primary database. Examples of each are described in the following sections.

Creating Standby from Active Target

Figure 5-1 illustrates the steps involved with using the RMAN *DUPLICATE* command to create a standby database. In this scenario it's a single-instance primary database being duplicated to a single-instance standby database (no RAC or ASM involved). Pay particular attention to Step 17, which is running the *DUPLICATE* command. If you were manually creating the standby database you would have to replace that step with several steps, such as "create an RMAN backup, create a standby control file, copy files to standby server, and restore and recover the standby database."

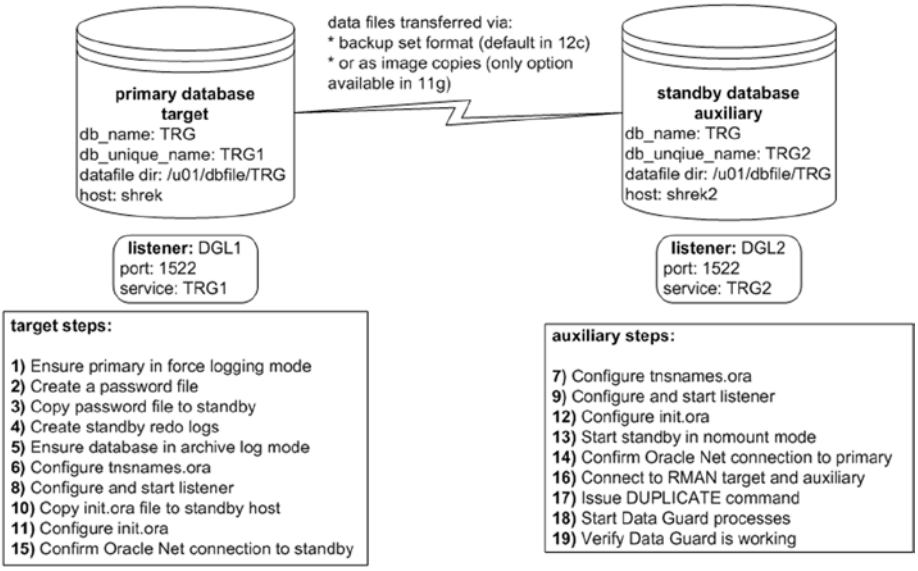

Figure 5-1. *Creating a standby database with RMAN active duplication*

A detailed description of using the *DUPLICATE* command to implement a standby database is next. As a prerequisite, ensure that the same version of Oracle that is installed on the primary database server is also installed on the standby database server. Also make sure all directories that contain the control files, data files, online redo logs, and archive redo logs exist on the standby server. Additionally, for active duplication there must be Oracle Net connectivity between the two hosts.

■ **Tip** If you're not familiar with Oracle Net, now would be a good time to review Chapter 6 in its entirety.

---

1. Ensure the primary target database is in force logging mode:

```
$ sqlplus / as sysdba
SQL> alter database force logging;
```

Verify that force logging has been set correctly, as follows:

```
SQL> select force_logging from v$database;

FORCE_LOGGING
---------------------------------------
YES
```

2. From the operating system, create a password file on the primary host. You'll have to modify this to match the database name used in your environment. The primary database name is *TRG* in this example:

```
$ cd $ORACLE_HOME/dbs
$ orapwd file=orapwTRG password=foo
```

3. Copy the primary database password file to the standby server. This example uses the Linux/UNIX *scp* command and is initiated from the target server:

```
$ cd $ORACLE_HOME/dbs
$ scp orapwTRG oracle@shrek2:$ORACLE_HOME/dbs
```

4. On the primary target database, create standby redo logs. Make very sure that the size of the standby redo logs is the exact same size as the existing online redo logs:

```
$ sqlplus / as sysdba
```

Here I verify the size of the existing online redo log files:

```
SQL> select distinct bytes from v$log;
     BYTES
----------
  52428800
```

Now I add the standby redo logs:

```
SQL> alter database add standby logfile
     '/u01/oraredo/TRG/sb1.rdo' size 52428800;
SQL> alter database add standby logfile
     '/u01/oraredo/TRG/sb2.rdo' size 52428800;
```

```
SQL> alter database add standby logfile
     '/u01/oraredo/TRG/sb3.rdo' size 52428800;
SQL> alter database add standby logfile
     '/u01/oraredo/TRG/sb4.rdo' size 52428800;
```

Next, verify the standby redo logs were created:

```
SQL> SELECT GROUP#,THREAD#,SEQUENCE#,ARCHIVED,STATUS FROM V$STANDBY_LOG;
```

Here is some sample output:

```
    GROUP#    THREAD#  SEQUENCE# ARC STATUS
---------- ---------- ---------- --- ----------
         3          0          0 YES UNASSIGNED
         4          0          0 YES UNASSIGNED
         5          0          0 YES UNASSIGNED
         6          0          0 YES UNASSIGNED
```

5. Ensure the primary database is in archive log mode:

```
SQL> archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            /u01/arch/TRG
```

If the primary database is not in archive log mode, then enable it, as follows:

```
SQL> shutdown immediate;
SQL> startup mount;
SQL> alter database archivelog;
SQL> alter database open;
```

6. Configure the *tnsnames.ora* file on the primary host (usually located in the *ORACLE_HOME/network/admin* directory):

```
TRG1 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1522))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = TRG1)))

TRG2 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shrek2)(PORT = 1522))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = TRG2)))
```

7. Configure the *tnsnames.ora* file on the standby host (usually located in the *ORACLE_HOME/network/admin* directory):

```
TRG1 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1522))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = TRG1)))

TRG2 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shrek2)(PORT = 1522))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = TRG2)))
```

8. Configure the listener on the primary and place the following in the *listener.ora* file (usually located in the *ORACLE_HOME/network/admin* directory):

```
DGL1 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1522))))

SID_LIST_DGL1 =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = TRG1) # static registration of TRG1 service
      (ORACLE_HOME = /orahome/app/oracle/product/12.1.0.2/db_1)
      (SID_NAME = TRG)))
```

Start the listener on the primary:

```
$ lsnrctl start DGL1
```

You should see this in the output:

```
Service "TRG1" has 1 instance(s).
  Instance "TRG", status UNKNOWN, has 1 handler(s) for this service...
```

The *UNKNOWN* status indicates the *TRG1* service has been statically registered with the listener.

9. Configure the listener on the standby server and place the following in the *listener.ora* file (usually located in the *ORACLE_HOME/network/admin* directory):

```
DGL2 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = shrek2)(PORT = 1522))))

SID_LIST_DGL2 =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = TRG2) # static registration of TRG2 service
      (ORACLE_HOME = /orahome/app/oracle/product/12.1.0.2/db_1)
      (SID_NAME = TRG)))
```

Start the listener on the standby host:

```
$ lsnrctl start DGL2
```

You should see this in the output:

```
Service "TRG2" has 1 instance(s).
  Instance "TRG", status UNKNOWN, has 1 handler(s) for this service...
```

This output indicates that the *TRG2* service has been registered statically with the listener. We know that the service was registered statically because of the *UNKNOWN* status.

10. Before modifying the primary database initialization file, first copy the primary database *init.ora* file to the standby database. This initiates the *scp* command from the primary database host:

```
$ scp $ORACLE_HOME/dbs/initTRG.ora oracle@shrek2:$ORACLE_HOME/dbs
```

11. Now add primary database–related initialization parameters in *init.ora* file (usually located in the *ORACLE_HOME/dbs* directory):

```
# Standby params
db_unique_name=TRG1
log_archive_config='DG_CONFIG=(TRG1,TRG2)'
log_archive_max_processes='6'
#
```

```
# Next two lines should be all on one line in initialization file:
log_archive_dest_1='LOCATION=/u01/arch/TRG valid_For=(all_
logfiles,all_roles)
  db_unique_name=TRG1'
#
# Next two lines should be all on one line in initialization
file:
log_archive_dest_2='SERVICE=TRG2 ASYNC
  valid_for=(online_logfiles,primary_role) db_unique_name=TRG2'
fal_server=TRG2
standby_file_management=auto
log_file_name_convert='/u01/oraredo/TRG','/u01/oraredo/TRG'
```

Stop and start the primary database to ensure the initialization parameters are instantiated:

```
SQL> startup force;
```

If you're not comfortable with running the prior command (which performs a *SHUTDOWN ABORT* and *STARTUP*), then stop and start the database as follows:

```
SQL> shutdown immediate;
SQL> startup;
```

12. Set the standby initialization parameters in the *init.ora* file (usually located in the *ORACLE_HOME/dbs* directory). In this example, the file is named *initTRG.ora* and has the following added to it:

```
# Standby params
db_unique_name=TRG2
log_archive_config='DG_CONFIG=(TRG1,TRG2)'
log_archive_max_processes='6'
#
# Next two lines should be all on one line in initialization file:
log_archive_dest_1='LOCATION=/u01/arch/TRG valid_For=(all_
logfiles,all_roles)
  db_unique_name=TRG2'
#
# Next two lines should be all on one line in initialization
file:
log_archive_dest_2='SERVICE=TRG1 ASYNC
  valid_for=(online_logfiles,primary_role) db_unique_name=TRG1'
fal_server=TRG1
standby_file_management=auto
log_file_name_convert='/u01/oraredo/TRG','/u01/oraredo/TRG'
```

13. Start up the standby in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
SQL> exit;
```

14. Ensure Oracle Net connectivity exists between the primary and standby servers. From the standby database server, ensure you can connect to the primary database over Oracle Net:

```
$ sqlplus sys/foo@TRG1 as sysdba
SQL> select host_name from v$instance;

HOST_NAME
----------------------------------------------------------
shrek
```

15. From the primary database server, ensure that you can connect to the standby database over Oracle Net:

```
$ sqlplus sys/foo@TRG2 as sysdba
SQL> select host_name from v$instance;

HOST_NAME
----------------------------------------------------------
shrek2
```

16. Connect to RMAN with the target and auxiliary databases. In this example, the RMAN connection is initiated on the auxiliary standby server (but could just as easily be initiated from the target server or a remote RMAN client):

```
$ rman target sys/foo@TRG1 auxiliary sys/foo@TRG2
```

In the prior line of code, you need to specify a net service name for the auxiliary database connection, otherwise you'll receive an error similar to this:

```
RMAN-06217: not connected to auxiliary database with a net service name
```

17. Issue the *DUPLCIATE* command to create a standby database on the auxiliary host:

```
RMAN> DUPLICATE TARGET DATABASE
FOR STANDBY
FROM ACTIVE DATABASE
DORECOVER
NOFILENAMECHECK;
```

Here's a small snippet of the lengthy output:

```
Starting Duplicate Db at
using target database control file instead of recovery catalog
allocated channel: ORA_AUX_DISK_1
channel ORA_AUX_DISK_1: SID=22 device type=DISK
contents of Memory Script:
...
Finished recover at
Finished Duplicate Db at
```

18. After the standby database is created you can start the Data Guard recovery process:

```
$ sqlplus / as sysdba
SQL> startup force;
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

In the prior code, if you're not comfortable using *STARTUP FORCE* (which does a *SHUTDOWN ABORT* and *STARTUP*), then instead issue *SHUTDOWN IMMEDATE* and then *STARTUP*.

One caveat is that starting with Oracle 12c, the *USING CURRENT LOGFILE* clause is no longer required when starting the Data Guard managed recovery process. If you're using Oracle 11g, the prior *ALTER DATABASE* command looks like this:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
USING CURRENT LOGFILE DISCONNECT;
```

19. Lastly, verify that Data Guard is working. You can verify the standby is operational in a number of ways. One simple way is to insert a record into a table on the primary and then query the table from the standby to see if the transactions are being applied to the standby. This assumes that you're using the Enterprise Edition of Oracle and Active Data Guard (this option requires a license from Oracle). If you're using standby redo logs, such as in this example, then the transactions should be transmitted from the primary database to the standby in near real time (seconds or less). You can also run the following query on the standby database to verify that the standby is applying transactions properly:

```
SQL> SELECT client_process, process, thread#, sequence#, status
FROM v$managed_standby
WHERE client_process='LGWR' OR PROCESS='MRP0';
```

The output indicates that Data Guard is alive and applying a log:

```
CLIENT_P PROCESS      THREAD#  SEQUENCE# STATUS
-------- --------- ---------- ---------- ------------
LGWR     RFS                1        283 IDLE
N/A      MRP0               1        283 APPLYING_LOG
```

An additional sanity check would be to switch the log file on the primary database and then check the *alert.log* on the standby to see if those logs are applied to the standby database. If Data Guard is working correctly you should see messages similar to this in the *alert.log*:

```
Media Recovery Waiting for thread 1 sequence 14 (in transit)
Recovery of Online Redo Log: Thread 1 Group 3 Seq 14 Reading mem 0
  Mem# 0: /u01/oraredo/TRG/sb1.rdo
```

Creating Standby from RMAN Backup

Creating a standby database from an RMAN backup is similar in many ways to the steps covered in the previous section, "Creating a Standby from Active Target." The main difference is that instead of duplicating from the active primary (target) database, you must instead first create an RMAN backup and copy it to the standby (auxiliary) server (or to storage that the standby database can access).

I won't repeat all of the steps regarding setting up a standby. Make sure you complete Steps 1 through 16 from the previous section in this chapter before starting here. Next is a description of the additional tasks required when creating a standby database from an RMAN backup.

1. First, create an RMAN backup on the primary database:

   ```
   $ rman target /
   RMAN> backup database plus archivelog;
   RMAN> exit;
   ```

2. Next, copy the backup files to the standby server. For my environment the backup files are in the */u01/rman/TRG* directory. This example uses the Linux/UNIX *scp* command and initiates the operation from the primary server:

   ```
   $ scp /u01/rman/TRG/*.* oracle@shrek2:/u01/rman/TRG
   ```

3. On the standby server, start up the auxiliary database in nomount mode:

   ```
   $ sqlplus / as sysdba
   SQL> startup nomount;
   SQL> exit;
   ```

4. On the standby server, connect to RMAN with both the target and auxiliary connections specified:

```
$ rman target sys/foo@TRG1 auxiliary sys/foo@TRG2
```

5. Assuming there's an RMAN backup available, issue the *DUPLICATE* command as follows:

```
RMAN> DUPLICATE TARGET DATABASE
FOR STANDBY
DORECOVER
NOFILENAMECHECK;
```

Notice that the prior *DUPLICATE* command in this scenario does not use the *FROM ACTIVE DATABASE* clause. It assumes there is an RMAN backup available. It knows where the RMAN backup is located based on the metadata in the target database control file, hence the need to connect to the target database (or a recovery catalog). If there is no RMAN backup available, you'll receive this error:

```
RMAN-06024: no backup or copy of the control file found to restore
```

You may receive an error similar to this when RMAN completes the duplication procedure:

```
ORA-01152: file 1 was not restored from a sufficiently old backup
ORA-01110: data file 1: '/ora01/dbfile/TRG/system01.dbf'
ORA-1547 signalled during: alter database recover cancel...
```

These error messages aren't as bad as they look. Perform these additional steps from SQL*Plus (on the auxiliary database). First, shutdown the database, start it in mount mode, and start the Data Guard processes, as follows:

```
$ sqlplus / as sysdba
SQL> shutdown immediate;
SQL> startup mount;
SQL> alter database recover managed standby database
disconnect from session;
```

That should apply the redo necessary to be able to open the database. You'll need to inspect the *alert.log* file of the auxiliary database and verify that the required archive redo logs have been successfully applied. If you're unsure of where the *alert.log* is located, run this query:

```
SQL> select value from v$diag_info where name='Diag Trace';
```

Here's the output for my standby database:

```
VALUE
------------------------------------------------------------
/orahome/app/oracle/diag/rdbms/trg2/TRG/trace
```

You should messages like this in the *alert.log*:

```
Media Recovery Log /u01/arch/TRG/TRG1_18_869840124.arc
Media Recovery Log /u01/arch/TRG/TRG1_19_869840124.arc
Recovery of Online Redo Log: Thread 1 Group 4 Seq 21 Reading mem 0
Mem# 0: /u01/oraredo/TRG/sb2.rdo
```

After the required archive redo logs have been applied, shut down the database and restart the standby database and the Data Guard processes:

```
SQL> shutdown immediate;
SQL> startup;
SQL> alter database recover managed standby database
disconnect from session;
```

This command works in Oracle 12c; if you're using Oracle 11g, you should run the following command instead to start the Data Guard processes:

```
SQL> alter database recover managed standby database
using current logfile disconnect;
```

■ **Tip** See MOS note 469493.1 for additional details on how to manually create a standby database.

One last note: If you're creating a standby database that is quite large in size (terabytes), you may want to encapsulate the *DUPLICATE* command within a shell script. Here's a typical Bash shell script that contains such code:

```
#!/bin/bash
date
export ORACLE_SID=TRG
export NLS_DATE_FORMAT='dd-mon-rrrr hh24:mi:ss'
#
sqlplus -s /nolog <<EOF
connect / as sysdba;
startup nomount;
select host_name from v\$instance;
exit;
EOF
```

```
#---------------------------------
rman <<EOF
connect target    sys/foo@TRG1
connect auxiliary sys/foo@TRG2
DUPLICATE TARGET DATABASE
FOR STANDBY
DORECOVER
NOFILENAMECHECK;
EOF
date
exit 0
```

Now, assuming the prior code is in a file named *dup.bsh*, you can make it executable, as follows:

```
$ chmod +x dup.bsh
```

When creating standby databases that are large in size, where it may take hours or even days to build the standby database, I'll usually run the duplication shell script in the background as follows:

```
$ nohup dup.bsh &
```

Running a script in the background like this serves two purposes. First, it creates an operating system file (named *nohup.out*) that captures the output of any commands executing within the shell script. It also allows me to continuously tail the output the shell script is producing:

```
$ tail -f nohup.out
```

If you work on servers that have an automatic timeout for idle sessions, running a shell script in the background is highly desirable, as the shell script will continue to execute even if your terminal session is automatically logged off from a timeout setting on the server.

Container and Pluggable Databases

The Oracle Multitenant option is available in Oracle 12c and higher. This feature allows you to create and maintain many pluggable databases within an overarching multitenant container database. A multitenant container database is defined as a database capable of housing one or more pluggable databases. A container is defined as a collection of data files and metadata that exist within a container database. A pluggable database is a special type of container that can be easily provisioned by cloning another database. If need be, a pluggable database can also be transferred from one container database to another.

Every container database contains a master set of data files and metadata known as the root container. Each container database also contains a seed container, which is used as a template for creating other pluggable databases. Each container database consists of one master root container, one seed container, and zero or one or more pluggable databases.

When working with containers and pluggable databases, there are two types of duplication you may require:

- Duplicating the entire container database (and all of the associated pluggable databases)
- Duplicating a subset of pluggable databases

Examples of each are shown in the following sections.

Duplicating a Container Database

It's fairly simple to duplicate an entire container database. Just like with a regular (non-container) database, you can either duplicate from an active target database or from RMAN backups. The main aspect to note about this scenario is that the initialization parameter file on the auxiliary must contain this line:

```
enable_pluggable_database=true
```

Also, you must ensure that you make a connection to the root container as *SYS* when performing the duplicate operation. Figure 5-2 illustrates the steps required to duplicate a container database. Notice that there must be a listener running on the target server (if you're unsure of how to start a listener and register services, see Chapter 6 for details).

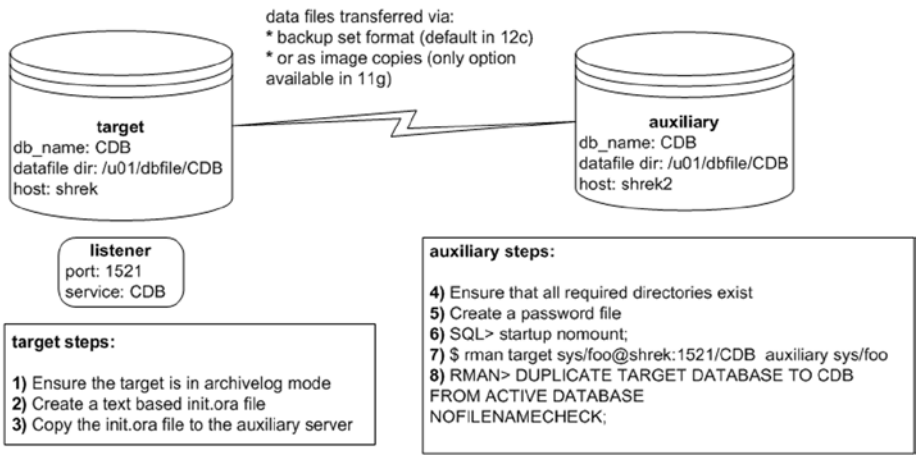

Figure 5-2. *Duplicating a container database*

These details of the duplication steps in this scenario are described next. This example assumes that a fully functioning container target database has already been created.

■ **Tip** See *Pro Oracle Database 12c Administration* available from Apress for details on creating and managing container and pluggable databases.

In this example, the target container database name is *CDB*. Additionally, the auxiliary database name and directory structure are identical to the target database.

1. Ensure the container target database is in archive log mode. Connect to the root container as a user with the *SYSDBA* privilege:

```
$ sqlplus / as sysdba
SQL> archive log list;
```

If it's not in archivelog mode, enable it as follows:

```
SQL> shutdown immediate;
SQL> startup mount;
SQL> alter database archivelog;
SQL> alter database open;
```

2. On the target, create a text-based initialization file if one doesn't already exist:

```
SQL> create pfile from spfile;
```

3. Copy the *initCDB.ora* file to the auxiliary server. This example uses the Linux/UNIX *scp* command and initiates the copy operation from the target database:

```
$ scp $ORACLE_HOME/dbs/initCDB.ora oracle@shrek2:$ORACLE_HOME/dbs
```

4. On the auxiliary server, ensure that all required directories exist to hold the data files, control files, and online redo logs. You can verify the directories in use by running this query on the target database:

```
SQL> select name from v$datafile
union
select name from v$tempfile
union
select name from v$controlfile
union
select member from v$logfile;
```

5. On the auxiliary server, create a password file for the *CDB* database:

```
$ cd $ORACLE_HOME/dbs
$ orapwd file=orapwCDB password=foo
```

6. On the auxiliary server, start up the *CDB* auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
```

At this point, it doesn't hurt to verify the *enable_pluggable_database* parameter is set to *TRUE*:

```
SQL> show parameter enable_pluggable_database
```

Here is some sample output:

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- -----------------
enable_pluggable_database            boolean     TRUE
```

7. On the auxiliary server, connect via RMAN to the target and auxiliary databases:

```
$ rman target sys/foo@shrek:1521/CDB  auxiliary sys/foo
```

You'll have to change the prior username, password, and connection information to match your environment.

8. Now execute the *DUPLICATE* command:

```
RMAN> DUPLICATE TARGET DATABASE TO CDB
FROM ACTIVE DATABASE
NOFILENAMECHECK;
```

You should now see some rather verbose output; here's a snippet:

```
executing Memory Script
...
database opened
Finished Duplicate Db at ...
```

When finished you should have an identical copy of the container database (and all of its associated pluggable databases). You can now connect to the root container and verify the container and pluggable databases exist via the following:

```
SQL> select con_id, name from v$containers;
```

Here's some sample output:

```
    CON_ID NAME
---------- ------------------------------
         1 CDB$ROOT
         2 PDB$SEED
```

Duplicating Pluggable Databases

You can use RMAN to duplicate one or more pluggable databases to an auxiliary database. When you duplicate a pluggable database, RMAN duplicates the container database that houses the pluggable databases and then also duplicates the pluggable databases that you choose.

You can either specify which pluggable databases you want to include or state which pluggable databases you want to exclude. Examples of each operation are described in the following subsections.

Including Pluggable Databases

Suppose you have two pluggable databases named *SALESPDB* and *HRPDB* that you require to be duplicated to a different environment. To achieve this, first follow steps 1 through 7 from the prior section, "Duplicating a Container Database." Make sure you create all of the required directories on the auxiliary server before you run the step containing the *DUPLICATE* operation. This means you must create all directories (on the auxiliary server) to house the pluggable database data files (as they existed on the target server).

Now, change the *DUPLICATE* command as follows to only include the pluggable databases of interest:

```
RMAN> DUPLICATE TARGET DATABASE TO CDB
PLUGGABLE DATABASE salespdb, hrpdb
FROM ACTIVE DATABASE
NOFILENAMECHECK;
```

You should see a great deal of output; here's a small sample for this example:

```
Starting Duplicate Db at ...
using target database control file instead of recovery catalog
allocated channel: ORA_AUX_DISK_1
...
```

```
sql statement: alter pluggable database all open
Dropping offline and skipped tablespaces
...
```

When the operation is finished, you can verify the pluggable databases that were duplicated:

```
$ sqlplus / as sysdba
SQL> select con_id, name from v$containers;
```

Here's some sample output confirming that only the pluggable databases of interest were duplicated:

```
    CON_ID NAME
---------- ------------------------------
         1 CDB$ROOT
         2 PDB$SEED
         3 SALESPDB
         4 HRPDB
```

Keep in mind that the duplicate operation in this section duplicates the entire container environment and additionally the pluggable databases specified in the *DUPLICATE* command. That's why the *CDB$ROOT* and *PDB$SEED* containers were also duplicated, as these are part of any base container database.

■ **Note** You can also instruct RMAN to only duplicate specific tablespaces within a PDB via the following syntax: *TABLESPACE pdb:tablespace*

Excluding Pluggable Databases

The other technique to duplicate specific pluggable databases is by specifying which pluggable databases you do *not* want duplicated. To achieve this, first follow steps 1 through 7 from the section "Duplicating a Container Database." Ensure you create all of the required directories on the auxiliary server before you run the step with the *DUPLICATE* operation. This means you must create all directories (on the auxiliary server) to house the pluggable database data files (as they existed on the target server).

For the last step, change the *DUPLICATE* command as follows to exclude pluggable databases that you don't want replicated. This example excludes the *HRPDB* pluggable database:

```
RMAN> DUPLICATE TARGET DATABASE TO CDB
SKIP PLUGGABLE DATABASE hrpdb
FROM ACTIVE DATABASE
NOFILENAMECHECK;
```

When finished you can verify that the container database contains the pluggable databases of interest:

```
$ sqlplus / as sysdba
SQL> select con_id, name from v$containers;
```

Here is some sample output:

```
    CON_ID NAME
---------- ------------------------------
         1 CDB$ROOT
         2 PDB$SEED
         3 SALESPDB
```

■ **Note** You can also instruct RMAN to only skip specific tablespaces within a PDB via the following syntax: *SKIP TABLESPACE pdb:tablespace*

RAC Databases

If you work in an environment that contains a mixture of RAC and non-RAC databases and find yourself with requirements to duplicate environments within this infrastructure, then you'll find the *DUPLICATE* command seamlessly solves many technical issues. Because of the complexity of RAC environments, this section doesn't document every detail involved with this operation, but it does give you a good high-level overview of a couple of common duplication scenarios involving RAC-enabled databases. Let's first take the case of duplicating a non-RAC environment to a RAC environment.

Non-RAC/Non-ASM to RAC/ASM

I was recently asked by my manager to create a copy of a 10 TB database that was non-RAC and non-ASM and create an RAC database that uses ASM storage. One caveat was that on the auxiliary (destination RAC and ASM database) there was only about 10 TB of storage, so there was not enough space to copy an RMAN backup (even if it was compressed) to the auxiliary server and restore the database.

In this situation, I decided to use the RMAN *DUPLICATE* command to replicate from the live target database. RMAN nicely handles the conversion of taking data files on regular storage and restoring them on ASM disks. Figure 5-3 depicts this scenario. This example assumes that the RAC grid/cluster software and Oracle database software has been installed in the auxiliary environment. It's beyond the scope of this book to document the steps to install this software, so here you'll need to look at Oracle's RAC and grid infrastructure software documentation, which is available freely on Oracle Technology Network website (*otn.oracle.com*).

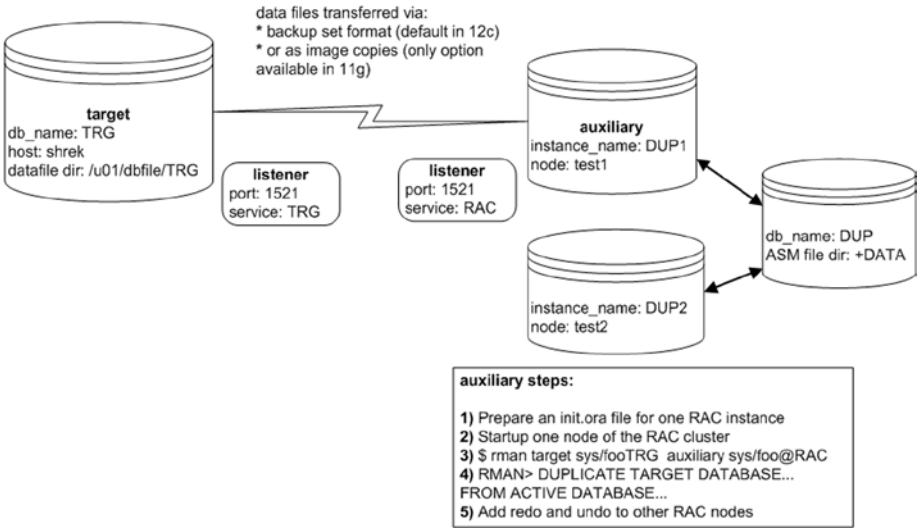

Figure 5-3. *Duplicating from non-RAC to RAC*

The steps depicted in figure 5-3 are described in detail next.

1. Prepare an *init.ora* file for one RAC instance on one RAC node. Ensure you initially set the *cluster_database* parameter to *FALSE*.

2. Start up one node of the RAC cluster:

```
$ sqlplus / as sysdba
SQL> startup nomount;
```

3. Connect to the target and auxiliary instances:

```
$ rman target sys/foo@shrek:1521/TRG auxiliary sys/foo@RAC
```

4. Issue the *DUPLICATE* command:

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
FROM ACTIVE DATABASE
  db_file_name_convert '/u01/dbfile/TRG', '+DATA/DUP/datafile'
logfile group 1
  ('+DATA/DUP/onlinelog/redot1g1m1.rdo',
   '+DATA/DUP/onlinelog/redot1g1m2.rdo') size 1073741824,
group 2
  ('+DATA/DUP/onlinelog/redot1g2m1.rdo',
   '+DATA/DUP/onlinelog/redot1g2m2.rdo') size 1073741824,
```

```
group 3
  ('+DATA/DUP/onlinelog/redot1g3m1.rdo',
   '+DATA/DUP/onlinelog/redot1g3m2.rdo') size 1073741824
NOFILENAMECHECK;
```

5. After the duplication procedure successfully completes, add any additional threads of *REDO* and *UNDO* to the other RAC nodes:

```
SQL> ALTER DATABASE ADD LOGFILE THREAD 2
  GROUP 10
   ('+DATA/DUP/onlinelog/redot2g10m1.rdo',
    '+DATA/DUP/onlinelog/redot2g10m2.rdo') SIZE 1073741824,
  GROUP 11
   ('+DATA/DUP/onlinelog/redot2g11m1.rdo',
    '+DATA/DUP/onlinelog/redot2g11m2.rdo') SIZE 1073741824,
  GROUP 12
   ('+DATA/DUP/onlinelog/redot2g12m1.rdo',
    '+DATA/DUP/onlinelog/redot2g12m2.rdo') SIZE 1073741824;
```

Enable the second thread of redo:

```
SQL> alter database enable public thread 2;
```

And now create the *UNDO* tablespace for the second node:

```
SQL> create undo tablespace undotbs2
datafile '+DATA/DUP/datafile/undotbs02_01.dbf' size 1g;
```

Also set *cluster_database=TRUE* in the initialization file, and you should now be able to start the second node.

RAC/ASM to Non-RAC/Non-ASM

Duplicating from a multinode RAC database using ASM storage to a single-instance database using the regular file system storage is fairly easy. The RMAN *DUPLICATE* command seamlessly handles the conversion from ASM-based storage to regular file system disks. This scenario is shown in Figure 5-4.

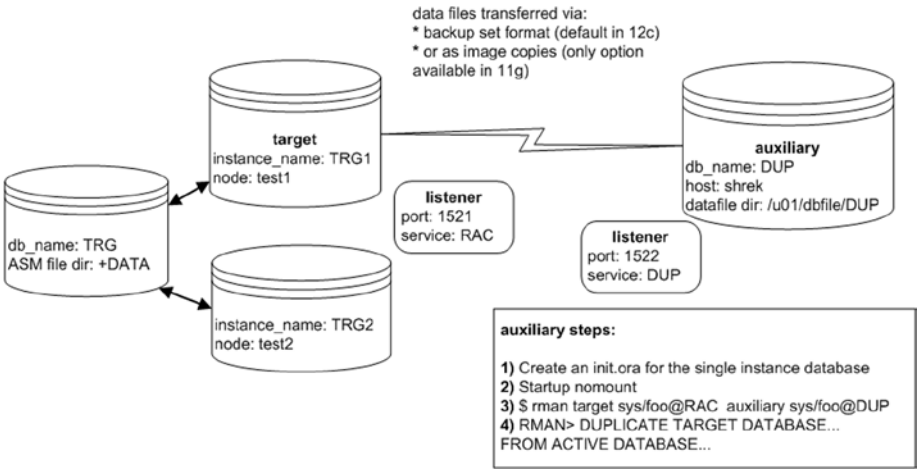

Figure 5-4. *RAC to non-RAC duplication*

The steps shown in Figure 5-4 are described next.

1. Create an *init.ora* file for a single-instance database.

Here's the *initDUP.ora* file I used for this example:

```
control_files='/u01/dbfile/DUP/control01.ctl'
db_block_size=8192
db_name='DUP'
fast_start_mttr_target=500
job_queue_processes=10
sga_max_size=400M
sga_target=400M
open_cursors=75
processes=100
remote_login_passwordfile='EXCLUSIVE'
resource_limit=true
standby_file_management='auto'
undo_management='AUTO'
undo_tablespace='UNDOTBS1'
workarea_size_policy='AUTO'
```

2. Start up the auxiliary database in nomount mode:

```
$ sqlplus / as sysdba
SQL> startup nomount;
```

3. Connect via RMAN to the target and auxiliary databases:

```
$ rman target sys/foo@RAC  auxiliary sys/foo@DUP
```

4. Issue the RMAN *DUPLICATE* command:

```
RMAN> DUPLICATE TARGET DATABASE TO DUP
FROM ACTIVE DATABASE
DB_FILE_NAME_CONVERT '+DATA/rac/datafile','/u01/dbfile/DUP',
                     '+DATA/rac/tempfile','/u01/dbfile/DUP'
LOGFILE GROUP 1 ('/u01/oraredo/DUP/redo01.rdo') SIZE 50M,
        GROUP 2 ('/u01/oraredo/DUP/redo02.rdo') SIZE 50M;
```

In the previous command, I used the *DB_FILE_NAME* conversion to ensure that the data files were mapped properly from the ASM storage to the regular file system storage.

If you see an error such as this:

```
ORA-38856: cannot mark instance UNNAMED_INSTANCE_2...
```

then place this line in your *init.ora* file:

```
_no_recovery_through_resetlogs=true
```

You should now be able to:

```
SQL> shutdown immediate;
SQL> startup mount;
SQL> alter database open resetlogs;
```

Summary

This chapter covered many additional scenarios with RMAN duplication. First, we investigated examples of partial database duplication. Then we then discussed increasing the performance through parallelism. When working with large databases, increasing the degree of parallelism can greatly improve performance.

Next, we discussed how to create Data Guard standby databases using the RMAN *DUPLICATE* command. You can create a standby database from a live target primary database or from an RMAN backup of a target primary database.

Duplicating container and pluggable databases was also covered. Duplicating an entire container database is very similar to duplicating a non-container database. You also have the option of duplicating a subset of pluggable databases housed within the container database.

Lastly, we discussed how to duplicate RAC/ASM databases to non-RAC/non-ASM databases and vice versa. The RMAN *DUPLICATE* command greatly assists in duplicating in RAC environments.

CHAPTER 6

Oracle Net Primer

This chapter describes the basics of Oracle Net. When using RMAN with active duplication, or to duplicate to a standby database, it's critical to understand how to configure and use Oracle Net. The examples described shortly deal with Oracle Net in a single-instance (non-RAC) and non-Oracle Restart environment; in other words, a single vanilla Oracle database installed on a host with nothing special configured. This chapter is not a comprehensive guide for Oracle's networking technology stack. Rather, it's an introduction to foundational Oracle Net material. I've found that a thorough understanding of the default behavior of Oracle Net in a single-instance environment is critical to successfully using RMAN to duplicate databases. Therefore, this chapter focuses on the following elementary topics:

- Detailing the default behavior of an Oracle Net service
- Creating and registering (dynamically and statically) services with the listener
- Controlling the listener

First, let's briefly discuss some background information on Oracle Net and its components.

Oracle Net

Oracle Net Services is the collection of technology that manages network connectivity between all components of the technology stack. *Oracle Net* is the component of Oracle Net Services that manages connectivity between remote clients and the Oracle database. The chief components of Oracle Net are:

Service: A service is an object created in the database that is used by Oracle to manage a connection to the database. Any connection to the Oracle database must be through a service. Many times the service is transparent to the client because either a default service is used during a connection or the service name is transparently used when the client uses an alias to resolve the connection information.

Listener: A process that resides on the database server that listens for incoming connections to the database.

lsnrctl executable: A utility that allows you to start, stop, reload, and view characteristics of the listener process. The *lsnrctl* utility can start the listener without any special settings. If you need more control over the listener behavior then you can place configuration settings in the *listener.ora* file.

listener.ora: This file controls the behavior of the listener process and is usually located in the *ORACLE_HOME/network/admin* directory. If you don't have a *listener.ora* file in place, then the listener uses default settings to operate. Some aspects you can control with this file are the name of the listener, the port used for listening for incoming connections, static registration of services with the listener, logging and tracing information, security settings, and so on.

Service registration: Before that service can be used for remote connections to the database, it must be registered with the listener. Registering a service with a listener gives the listener the ability to accept incoming connections via that service. The listener will refuse to connect any remote connections attempting to connect to a service that hasn't been registered. A service is registered with the listener either dynamically or statically.

LREG: The listener registration process. This process is responsible for dynamically registering services with listeners. The *PMON* process is responsible for registering services with the listener in Oracle 11g and previous editions. Whenever I mention *LREG* in this chapter, if you're working with Oracle 11g and prior, then you should substitute *PMON* for *LREG*. The main point here is that there is an Oracle background process that will automatically attempt to dynamically register services with the listener under certain conditions.

Dynamic registration: The *LREG* background process periodically wakes up and attempts to register any new instances (and associated services) running on the server with any listeners running on the server. By default *LREG* will only register services with listeners listening on the default port of 1521. If a listener is not listening on port 1521, then *LREG* will only automatically register an instance's services if the *LOCAL_LISTENER* initialization parameter has been defined with the port information a listener is listening for. You can manually instruct *LREG* to wake up and attempt service registration with the following:

```
SQL> alter system register;
```

Static registration: You can instruct the listener to perform registration for a service by placing information regarding the service in the *listener.ora* configuration file. This method is called *static* because the information in the *listener.ora* file is static (unless you change it). In other words, when you start a listener it will perform service registration for any service entries it finds that have been statically configured for the listener in the *listener.ora* file.

Remote client: A program that connects to a database over the network (e.g., SQL*Plus, RMAN, Java using JDBC, and so on).

Easy connection method: The name given to remote clients connecting over the network to a database where the remote client specifies the host name, the port the listener is listening on, and the service name directly in the string used to connect to the remote database. If you know the name of the host, port, and service name, you can directly enter those on the command line. The syntax is as follows:

```
$ sqlplus user/pass@[//]host[:port][/service_name][:server][/instance_name]
```

For example, the following SQL*Plus connection specifies a host, a port, and the service name of the remote database being connected to:

```
$ sqlplus system/foo@shrek:1521/TRG
```

In the prior connection string the host is *shrek*, the port is 1521, and the service name is *TRG*.

Local naming method: The name given to clients connecting over the network to a database by using the *tnsnames.ora* file to provide information such as the host the database is running on, the port the listener is listening on, and the service name.

tnsnames.ora: This file maps a connection alias to a host name, a port the listener is listening on, and the service name. This file is used in the local naming connection method and is usually located in the *ORACLE_HOME/network/admin* directory. For example, say the following entry is placed in the *tnsnames.ora* file:

```
TRG =
  (DESCRIPTION =
    (ADDRESS=(PROTOCOL=TCP)(HOST=shrek)(PORT=1521))
    (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=TRG)))
```

Then a connection to the *TRG* service using the local naming method would be as:

```
$ sqlplus system/foo@TRG
```

The examples in this chapter will use either the easy connect naming method or the local naming method. If you require details on other naming methods (e.g., directory naming method, like Oracle Internet Directory) that Oracle supports, see the *Oracle Database Net Services Administrator's Guide* freely available for download on Oracle's technology website (*otn.oracle.com*).

Now that you have an understanding of the basic components of Oracle Net, let's dive deeper into how to use a service to connect remotely to a database. One key aspect to keep in mind when reviewing the following material is that a service, a listener, and how a service gets registered with a listener are three separate concepts.

Services Default Behavior

It's critical to understand the default behavior of a service within an Oracle database. This lays the foundation for understanding how Oracle Net functions. When you create a new database there will normally be three default services created for you. For reference, here's the SQL script I used to create a basic single-instance Oracle database:

```
CREATE DATABASE TRG
   MAXLOGFILES 16
   MAXLOGMEMBERS 4
   MAXDATAFILES 1024
   MAXINSTANCES 1
```

```
   MAXLOGHISTORY 680
   CHARACTER SET AL32UTF8
DATAFILE
'/u01/dbfile/TRG/system01.dbf' SIZE 500M REUSE
   EXTENT MANAGEMENT LOCAL
UNDO TABLESPACE undotbs1 DATAFILE
'/u01/dbfile/TRG/undotbs01.dbf' SIZE 800M
SYSAUX DATAFILE
'/u01/dbfile/TRG/sysaux01.dbf' SIZE 500M
DEFAULT TEMPORARY TABLESPACE TEMP TEMPFILE
'/u01/dbfile/TRG/temp01.dbf' SIZE 500M
DEFAULT TABLESPACE USERS DATAFILE
'/u01/dbfile/TRG/users01.dbf' SIZE 20M
LOGFILE GROUP 1 ('/u01/oraredo/TRG/redo01a.rdo') SIZE 50M,
        GROUP 2 ('/u01/oraredo/TRG/redo02a.rdo') SIZE 50M
USER sys    IDENTIFIED BY foo
USER system IDENTIFIED BY foo;
--
@?/rdbms/admin/catalog.sql
@?/rdbms/admin/catproc.sql
conn system/foo
@?/sqlplus/admin/pupbld.sql
```

In this script, the name of the database is *TRG*. After the script finishes, we can display the services available in the *DBA_SERVICES* view thusly:

```
SQL> select service_id, name, network_name,
to_char(creation_date,'dd-mon-yy hh24:mi') create_date
from v$services
order by service_id;
```

As you can see, there are by default three services that have been created:

```
SERVICE_ID NAME           NETWORK_NAME CREATE_DATE
---------- -------------- ------------ ----------------
         1 SYS$BACKGROUND              25-jan-15 15:28
         2 SYS$USERS                   25-jan-15 15:28
         3 TRG            TRG          25-jan-15 15:28
```

The *SYS$BACKGROUND* service is the service that the Oracle background processes use when establishing a connection to the database. The *SYS$USERS* service is used for local connections. A third service is created that gets its name from the setting of the *SERVICE_NAMES* parameter. By default the *SERVICE_NAMES* parameter is populated by the value of *DB_UNIQUE_NAME*, and by default *DB_UNIQUE_NAME* is populated from the *DB_NAME* initialization parameter. In other words, for a freshly created Oracle database, if you don't set the *SERVICE_NAMES* or the *DB_UNIQUE_NAME* initialization

parameters, a service is created by default with the same name as the value in the *DB_NAME* initialization parameter:

```
SQL> show parameter db_name
NAME                                 TYPE        VALUE
------------------------------       ------      ------
db_name                              string      TRG
```

So we now have a service that can be used to connect remotely to the database. On a remote box, using the easy connection method, let's try to connect to the database through the *TRG* service:

```
$ sqlplus system/foo@shrek:1521/TRG
```

The connection fails with this error:

```
ORA-12541: TNS:no listener
```

A service by itself doesn't allow for remote connectivity from a client to a database; it's the combination of three components that does the trick:

- A service must exist.
- A listener process must be running on the database server (host).
- The service must be registered with the listener before remote connections can access the database through the given service.

So far in this example, the first item is true; a service named *TRG* does exist. We've verified that by viewing the information in the data dictionary. Take the second item from the prior list: Is a listener running? From the operating system on the database server, let's check the status of the listener:

```
$ lsnrctl status
```

This error is present in the output, thus indicating that no listener is running:

```
TNS-12541: TNS:no listener
```

You can start a listener on the database server without configuring any Oracle Net files. The listener works perfectly fine with default settings. So, without configuring anything, let's start the listener on the database server:

```
$ lsnrctl start
```

Here's a partial listing of the output:

```
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=shrek)(PORT=1521)))
The listener supports no services
The command completed successfully
```

We have a listener running now, and by default the listener is listening on port 1521 (as shown in the prior output). Let's again attempt to connect to the database from a remote client:

```
$ sqlplus system/foo@shrek:1521/TRG
```

Now this error is displayed:

```
ORA-12514: TNS:listener does not currently know of service requested in
connect descriptor
```

By default the *LREG* background process (*PMON* in Oracle 11g and before) periodically checks (every 60 seconds) to see if a new instance and listener are running on the server. If the listener is listening on port 1521, then the *LREG* process will automatically register any services contained in the *SERVICE_NAMES* parameter with the listener. It's been about a minute, so let's ask the listener to display services that are registered:

```
$ lsnrctl services

Service "TRG" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

The output indicates the service *TRG* has been registered with the instance *TRG*. By default a service is registered with the listener to listen for a particular instance. The instance in this name also happens to be *TRG*. In a single-instance database, the instance name is usually the same as the database name (the value in the *DB_NAME* parameter).

So, by default, you get a service, and once started the listener will by default listen on port 1521, and by default the *LREG* process will register services with a listener listening on port 1521. Let's try to connect remotely again:

```
$ sqlplus system/foo@shrek:1521/TRG
```

Bingo, all the pieces come together; the remote connection was successful:

```
SQL>
```

The prior connection details are depicted in Figure 6-1 and are described as follows. First, *LREG* registers the *TRG* service with the listener listening on port1521 on the *shrek* server (step 1). Second, a remote client uses the easy connection method to connect to the *TRG* service in the *TRG* database (step 2). Once successful, the listener hands off the connection to an Oracle server process (step 3). From that point on the Oracle server process handles all communication between the remote client and the database (step 4).

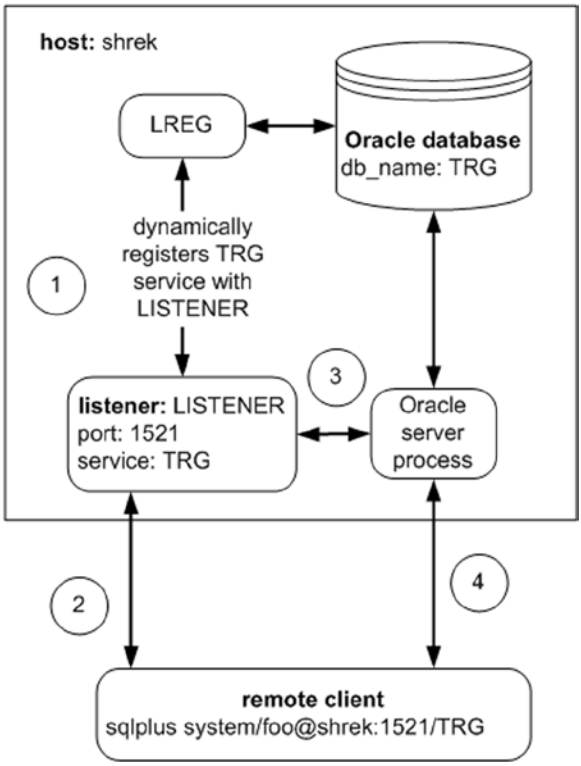

Figure 6-1. *A remote client connects to an Oracle database with default service and port*

By using default settings you can make your database available for use via remote connections. In the real world the use of default settings is not recommended, nor are they seldom used. Most hackers know that by default an Oracle listener listens on port 1521, so that port is where most hackers start when trying to remotely access a database. Having said that, I will use the default settings in several examples in this chapter, as it's important to understand how to identify default configurations and modify them appropriately.

It's worth mentioning at this point that you can modify the default service that gets created by setting the *DB_UNIQUE_NAME* initialization parameter. Setting *DB_UNIQUE_NAME* is a requirement when configuring Data Guard environments. When you set the value of *DB_UNIQUE_NAME* a service is created and named with the setting of *DB_UNIQUE_NAME*. To modify this parameter you can use the *ALTER SYSTEM* command. If you're using an SPFILE you must specify *SCOPE=SPFILE*, meaning that it cannot be set dynamically in memory:

```
SQL> alter system set db_unique_name='TRG1' scope=spfile;
```

Start and stop the instance in order for the value to take effect:

```
SQL> shutdown immediate;
SQL> startup;
```

Now, instead of a service with the name of *TRG* being registered with the listener, the service name should be *TRG1*:

```
$ lsnrctl services
```

After *LREG* has had a chance to dynamically register the service, the output verifies that the service *TRG1* in the instance *TRG* is registered with the listener:

```
Service "TRG1" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

Now that we've covered the default behavior of Oracle Net, let's look at non-default ways to create a service and register it with the listener.

Creating and Registering Services with the Listener

In most circumstances you'll require more from Oracle Net than just the default behavior. For example, you may want to register several services with the listener. Techniques to accomplish this are listed next:

- Set the *SERVICE_NAMES* initialization parameter to contain multiple service names and let *LREG* dynamically register the services (either by having the listener listen on port 1521 or by setting the *LOCAL_LISTENER* parameter to contain information regarding the port the listener is listening on).
- Static registration creates and registers a service for each service statically listed in the *listener.ora* file.
- The *DBMS_SERVICE* internal PL/SQL package allows you to create, activate, and drop services.

Each of these techniques is described in the following sections. First up is setting the *SERVICE_NAMES* initialization parameter.

Setting SERVICE_NAMES

One method for creating and registering multiple services with the listener is to populate the *SERVICE_NAMES* initialization parameter. Recall that if you don't specify a value for *SERVICE_NAMES* the default value is derived from *DB_UNIQUE_NAME*, and if *DB_UNIQUE_NAME* is not set, the default value is derived from *DB_NAME*. For this example, say that the *SERVICE_NAMES* parameter is not currently set to anything and has a default value of *TRG*, which it derived from the *DB_NAME* (which is *TRG* for my database).

In this example you have the requirement to add a service named *HRS* to the database. The following *ALTER SYSTEM* command sets the *SERVICE_NAMES* parameter to specify a *TRG* service and an *HRS* service:

```
SQL> alter system set service_names='TRG, HRS';
```

Querying *V$SERVICES* shows the service has been added to the instance:

```
SQL> select service_id, name, network_name,
to_char(creation_date,'dd-mon-yy hh24:mi') create_date
from v$services
order by service_id;
```

Here is the output:

```
SERVICE_ID NAME                 NETWORK_NAME         CREATE_DATE
---------- -------------        ------------         ---------------
         1 SYS$BACKGROUND                            25-jan-15 15:28
         2 SYS$USERS                                 25-jan-15 15:28
         3 TRG                  TRG                  25-jan-15 15:28
         4 HRS                  HRS                  25-jan-15 16:14
```

Recall that just because a service has been created doesn't mean that the service has also been registered with the listener. A service is only dynamically registered with the listener (by *LREG*) when:

- The listener is listening on port 1521
- The *LOCAL_LISTENER* initialization parameter contains information regarding the port the listener is listening on.

By default, if the listener is listening on port 1521, *LREG* will dynamically register services contained in the *SERVICE_NAMES* parameter with the listener.

To continue this example, assume there is a listener listening on port 1521. Let's observe any services registered with this listener:

```
$ lsnrctl services
```

Here's a snippet of the relevant output:

```
Service "HRS" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
...
Service "TRG" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

We can see that both the services, *TRG* and *HRS*, are registered with the listener, which is listening on port 1521. Let's see if we can connect remotely using the newly created and registered *HRS* service:

```
$ sqlplus system/foo@shrek:1521/HRS
SQL>
```

Yes, the connection was successful, as evidenced by the SQL prompt. Now what happens if we stop the listener and restart it by listening on the non-default port of 1529? First stop the currently running listener:

```
$ lsnrctl stop
```

In my environment, to configure the listener to listen on port 1529, I must navigate to the *ORACLE_HOME/network/admin* directory and place within the *listener.ora* file the following code:

```
LISTENER =
  (DESCRIPTION_LIST=
    (DESCRIPTION=
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1529))
       )
     )
   )
```

Next, start the listener:

```
$ lsnrctl start
```

The output indicates the listener is now listening on port 1529 and supports no services (meaning no services are registered with this listener):

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=shrek)(PORT=1529)))
The listener supports no services
The command completed successfully
```

With nothing else configured, this listener will never contain registered services. The *LREG* background process is currently only attempting to register services for a listener listening on port 1521, which now does not exist on this host. Notice what happens if we attempt to connect from a remote client to this database via a service:

```
$ sqlplus system/foo@shrek:1529/HRS
```

We receive a message indicating the service is not registered with the listener:

```
ORA-12514: TNS:listener does not currently know of service requested
in connect descriptor
```

How do we get *LREG* to register listeners running on the non-default port of 1529? There are two techniques:

- Set the *LOCAL_LISTENER* parameter to contain the host and port information.
- Set the *LOCAL_LISTENER* parameter to point to an entry in the *tnsnames.ora* file that contains the host, port, and service information.

Setting the LOCAL_LISTENER to contain host and port information

The first method involves setting the *LOCAL_LISTENER* initialization parameter to contain information regarding the host and port 1529. This instructs *LREG* to, instead of looking for listeners on port 1521, attempt to register services with a listener listening on port 1529. Use the *ALTER SYSTEM* command to set the *LOCAL_LISTENER* parameter to contain information regarding the host and port:

```
SQL> alter system set
local_listener='(ADDRESS=(PROTOCOL=TCP)(HOST=shrek)(PORT=1529))';
```

Now let's view the services registered with the listener:

```
$ lsnrctl services
```

Here's a partial listing of the output:

```
Service "HRS" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
...
Service "TRG" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

And we can verify that a remote connection is now possible via the *HRS* service:

```
$ sqlplus system/foo@shrek:1529/HRS
SQL>
```

The above connection process is shown in Figure 6-2 and described next. First, the initialization parameters *SERVICE_NAMES* and *LOCAL_LISTENER* are populated (step 1). Next, the *listener.ora* file is modified to contain the port 1529 (step 2). Next, the listener is stopped and started so that it is now listening on port 1529 (step 3). Subsequently, the LREG process dynamically registers the services *TRG* and *HRS* with the listener (step 4). Then the client process attempts a remote connection to the *HRS* service on port 1529 on the *shrek* server (step 5). When successful, the listener hands off the connection to an Oracle server process (step 6). From that point on the Oracle server process handles all communication between the database and client (step 7).

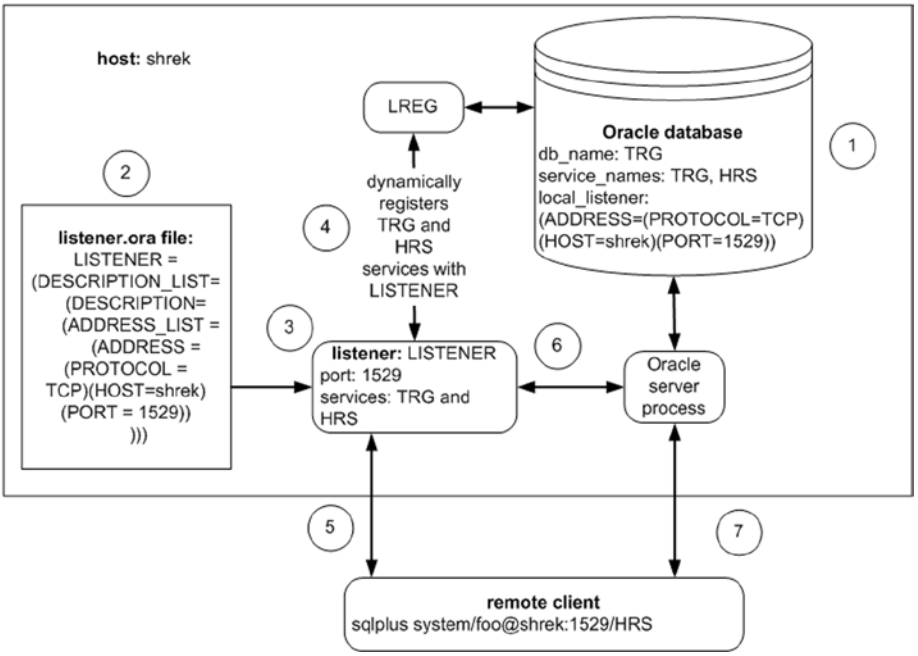

Figure 6-2. *Remote connection using a non-default port and non-default service*

Setting LOCAL_LISTENER to point to an entry in tnsnames.ora

There's another technique for setting the *LOCAL_LISTENER* parameter. This involves setting the value of *LOCAL_LISTENER* to what is effectively a pointer to an entry in the *tnsnames.ora* file. The entry in the *tnsnames.ora* file contains the host, port, and service information. Recall that the *tnsnames.ora* file is usually located in the *ORACLE_HOME/network/admin* directory. To demonstrate this technique, first place an entry in the *tnsnames.ora* file as follows:

```
LL=(ADDRESS=(PROTOCOL=TCP)(HOST=shrek)(PORT=1529))
```

Next, set the value of *LOCAL_LISTENER* to point to the entry in the *tnsnames.ora* file:

```
SQL> alter system set local_listener='LL';
```

Now let's confirm that the listener is listening on port 1529 for services defined in the *SERVICE_NAMES* parameter:

```
$ lsnrctl services

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=shrek)(PORT=1529)))
Services Summary...
```

```
Service "HRS" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
...
Service "TRG" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

And let's also confirm that we can connect from a remote client:

```
$ sqlplus system/foo@shrek:1529/HRS
SQL>
```

Statically Registering Services

The material thus far has outlined how to create and register multiple services with the listener using the *SERVICE_NAMES* parameter in combination with the default behavior of the listener listening on port 1521, and also with setting the listener to listen on a non-default port such as 1529.

This current section demonstrates how to create and register multiple services with a listener using static service registration. Static registration means the details of the service have been statically placed in the *listener.ora* file (the *listener.ora* file is static until you change it, anyway). For the material in this chapter covered up to this point, the *listener.ora* file only contains the following lines that specify which port the listener is listening on for a given host:

```
LISTENER =
  (DESCRIPTION_LIST=
    (DESCRIPTION=
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1529))
       )
     )
   )
```

For this static registration example, set the *LOCAL_LISTENER* parameter to contain a null value, meaning that no dynamic registration will take place:

```
SQL> alter system set local_listener='';
```

Also set *SERVICE_NAMES* to contain a null value:

```
SQL> alter system set service_names='';
```

When checking to see what services are registered with the listener we expect it to be none, because the listener is not listening on the default port of 1521 and we have not set the *LOCAL_LISTENER* parameter to point at port 1529:

```
$ lsnrctl services
```

Here's a partial listing of the output verifying our thinking that the listener is indeed running but with no registered services:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=shrek)(PORT=1529)))
The listener supports no services
```

To statically register a service with the listener you need to place an additional section in the *listener.ora* file that lists the services you require the listener to register. This addition to the *listener.ora* file is the *SID_LIST_LISTENER* section. The following lines of code are added to the *listener.ora* file specifying the static registration information for two services (*TRG* and *HRS*):

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = TRG) # service name
      (ORACLE_HOME = /u01/app/oracle/product/12.1.0.2/db_1)
      (SID_NAME = TRG) # instance name
     )
    (SID_DESC =
      (GLOBAL_DBNAME = HRS) # service name
      (ORACLE_HOME = /u01/app/oracle/product/12.1.0.2/db_1)
      (SID_NAME = TRG) # instance name
     )
  )
```

In this code, the *GLOBAL_DBNAME* specifies the service name, and *SID_NAME* is the instance name. So a service *TRG* is registered with the *TRG* instance, and the service *HRS* is registered with the *TRG* instance. Also note that any text on a line that appears after a # sign is a comment. Now stop and restart the listener (you could also do a reload; more on this later). When the listener is stopped and started:

```
$ lsnrctl stop
$ lsnrctl start
```

Here's part of the output it displays:

```
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=shrek)(PORT=1529)))
Services Summary...
Service "HRS" has 1 instance(s).
  Instance "TRG", status UNKNOWN, has 1 handler(s) for this service...
Service "TRG" has 1 instance(s).
  Instance "TRG", status UNKNOWN, has 1 handler(s) for this service...
```

The *UNKNOWN* status means the service has been statically registered with the listener. In other words, the status of *UNKNOWN* doesn't mean the listener isn't aware of the service; rather, it means that the service was not dynamically registered by *LREG*. So don't let the status of *UNKNOWN* confuse you, as it simply means the service was statically registered.

Now that the services have been statically registered with the listener, we can again connect remotely from a client:

```
$ sqlplus system/foo@shrek:1529/HRS
SQL>
```

The following query and output shows the successful connection:

```
SQL> select instance_name from v$instance;

INSTANCE_NAME
-------------
TRG
```

If you want to use the local naming method (and not easy connect), then add an entry to the *tnsnames.ora* file on the client machine, as follows:

```
HRS =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1529))
    (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = HRS)))
```

And now the client connection is initiated as follows:

```
$ sqlplus system/foo@HRS
```

The SQL*Plus utility knows from the connection string that local naming is being used and that it needs look in the *tnsnames.ora* file for details on the host, port, and service name.

The prior static registration and local naming connection method is depicted in Figure 6-3 and described here. First, the *listener.ora* file is modified to contain the static service information (step 1). Next, the listener is stopped and started so that it statically registers the services contained in the *listener.ora* file (step 2). Then the client *tnsnames.ora* file is modified to contain local naming information such as the host, port, and service name (step 3). Next, the remote client attempts to connect to the database (step 4). When successful, the listener hands off the connection to an Oracle server process (step 5). From that point on the Oracle server process handles all communication between the database and remote client (step 6).

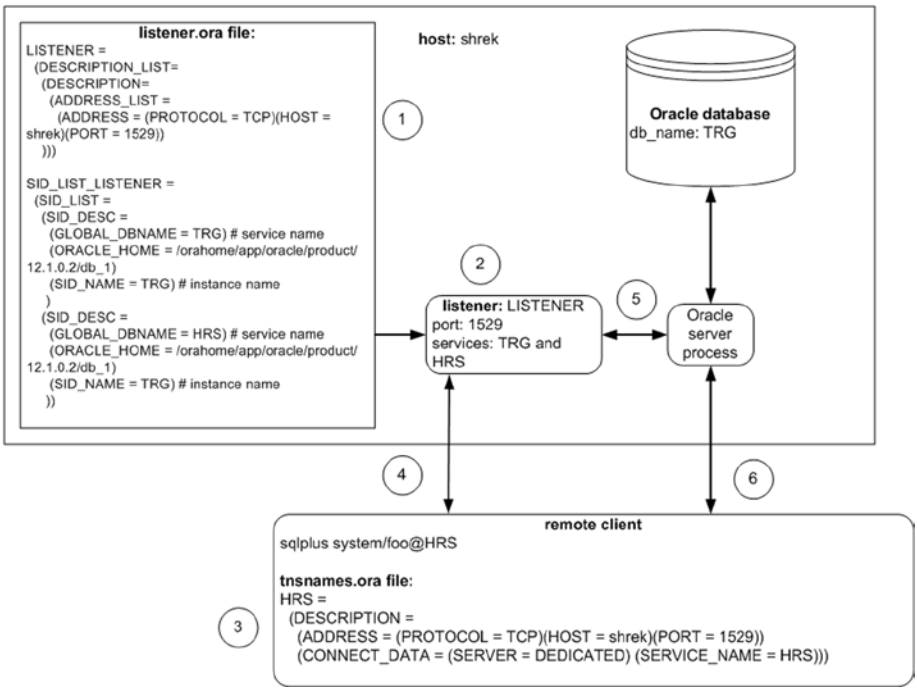

Figure 6-3. *Static registration and local naming connection*

Dynamic Registration versus Static Registration

The prior two sections detailed dynamic and static registration options. Which technique is preferable? Either dynamic or static registration is acceptable. There are some specific situations in which you may want to use static registration:

- If it's required by Oracle Enterprise Manager Cloud Control.
- If you want the service to be immediately available when the listener is started; static registration provides this.
- If you want the listener to automatically register any statically listed services when it starts or reloads.

Dynamic registration also has some advantages:

- You don't need to maintain information regarding the services in the *listener.ora* file.
- If need be you can instruct *LREG* to wake up and perform dynamic registration with the following:

```
SQL> alter system register;
```

This command instructs *LREG* to immediately register any services with a listener. This works if the listener is listening on the default port of 1521 or if you've configured the *LOCAL_LISTENER* parameter to include information regarding the host and port number that the listener is listening on.

Here are some other differences between dynamic registration and static registration that you should be aware of:

- When you stop an instance, dynamically registered services are no longer registered with the listener, whereas statically registered services still show up as registered with the listener (even though there is no instance running).
- It's also possible to statically register a service with a listener when there is no instance running.
- By default, statically registered services don't appear in the *V$SERVICES* or *DBA_SERVICES* views. You can change this behavior by using *DBMS_SERVICE* to create and start the statically registered service.
- Dynamically registered services do appear automatically in the *V$SERVICES* and *DBA_SERVICES* views.
- By default, when you connect to a statically registered service, the *SERVICE_NAME* column in *V$SESSION* shows the connection coming through the *SYS$USERS* service. You can change this behavior by using *DBMS_SERVICE* to create and start the statically registered service.

You may work with a group that has a strong opinion on which approach is best used (because of some of the differences listed previously), and that's fine.

Why Use Multiple Services?

You may also be asking why not just use the default service that's created when you create the database; why is it necessary to create multiple services? When you use a service to connect to the database Oracle tracks which service you use. Oracle populates various performance-related views with the service information along with resource usage per service. This allows you to monitor and report on performance metrics per each service. For example, you may have several applications using your database. You can create a service for each application to use when connecting to the database, which will allow you to monitor resource usage per the service connection.

Using DBMS_SERVICE

Oracle provides the built-in PL/SQL package *DBMS_SERVICE* to manage services. Normally you don't need to manage services with the degree of control provided through *DBMS_SERVICE*. However, you'll be better able to understand and maintain your Oracle

Net environment with a basic knowledge of how to use this package. There are several tasks you can perform with *DBMS_SERVICE*. The main features I'll cover here are:

- Adding and starting a service
- Stopping and deleting a service

Keep in mind there are other uses for *DBMS_SERVICE* (e.g., modifying a service, disconnecting a session, and so on). What I want to cover here are some of the features you'll use most frequently. Let's start with adding a service.

Adding a service

Suppose you want to use *DBMS_SERVICE* to add a service named *DW* to the instance. This next line of code demonstrates how to do so:

```
SQL> exec DBMS_SERVICE.CREATE_SERVICE(service_name=>'DW',
network_name=>'DW');
```

Checking in the *DBA_SERVICES* view we can confirm the creation of the service:

```
select service_id, name, network_name,
to_char(creation_date,'dd-mon-yy hh24:mi') create_date
from dba_services
order by service_id;
```

Here is the output:

```
SERVICE_ID NAME           NETWORK_NAME CREATE_DATE
---------- -------------- ------------ ---------------
         1 SYS$BACKGROUND              25-jan-15 15:28
         2 SYS$USERS                   25-jan-15 15:28
         3 TRG            TRG          25-jan-15 15:28
         4 HRS            HRS          25-jan-15 16:14
         5 DW             DW           25-jan-15 17:12
```

Now that the service has been created, and before it becomes available for use with remote connections, it must be registered with the listener. In this environment I'm using dynamic registration to register the service with the listener. I can force *LREG* to wake up and register any new services:

```
SQL> alter system register;
```

Assuming dynamic registration has been set up, when I look to see if the new service has been registered, it has not yet been:

```
$ lsnrctl services
Service "TRG" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

Before a service created with *DBMS_SERVICE* can be registered with the listener it must be started. Let's do that as follows:

```
SQL> exec DBMS_SERVICE.START_SERVICE(service_name=>'DW');
```

Now instruct *LREG* to wake up and register any started services:

```
SQL> alter system register;
```

The service is now registered with the listener:

```
$ lsnrctl services
...
Service "DW" has 1 instance(s).
  Instance "TRG", status READY, has 1 handler(s) for this service...
```

We can now connect remotely using this new service:

```
$ sqlplus system/foo@shrek:1529/DW
SQL>
```

Removing a service

Removing a service is the opposite of adding it. First, stop any services and then drop them. For example, to stop and delete the *DW* service do as follows:

```
SQL> exec DBMS_SERVICE.STOP_SERVICE(service_name=>'DW');
SQL> exec DBMS_SERVICE.DELETE_SERVICE(service_name=>'DW');
```

To wrap this section up, the *DBMS_SERVICE* package can be used to add and remove services. If you're using dynamic registration then the service will automatically be registered by *LREG* with the listener.

Displaying Service Information

When troubleshooting issues with remote connectivity you'll invariably want to determine which services are registered with the listener and which services exist in the database. You can display service information either through Oracle-provided operating system utilities or via data dictionary views.

LSNRCTL

The *lsnrctl* utility is the best source for showing which services are registered with the listener. For example:

```
$ lsnrctl services
```

The output will show details such as:

- Service names (and associated instance names) registered with the listener
- Whether service was registered dynamically (status of *READY*) or statically (status of *UNKNOWN*)
- Host name and port the listener is listening on

You can also display helpful status information regarding the listener as follows:

```
$ lsnrctl status
```

The *STATUS* option includes much of the same information the *SERVICES* option contains. In addition, the status displays how long the listener has been running as well as parameter and logging files.

Data dictionary views

There are also several data dictionary views available for looking at service information. The two primary views are *V$SERVICES* and *DBA_SERVICES*. These views display much of the same information, such as services that have been registered dynamically or that have been created and started via *DBMS_SERVICES*. The *V$SERVICES* view shows only services that have ever been active. The *DBA_SERVICES* shows all services that have ever been created regardless of whether they have ever been active.

You can view the service associated with a remote connection by querying the *SERVICE_NAME* column of the *V$SESSION* view. This is helpful for determining connections to the database and the corresponding services being used for the connections.

Other views such as *V$SERVICEMETRIC* and *V$SERVICEMETRIC_HISTORY* display resource usage for services. These views are useful for determining the usage and performance metrics for sessions using a particular service.

TNSPING

Oracle provides the *tnsping* utility to help troubleshoot Oracle Net connectivity issues between a client and a remote database. For example, if you want to verify the existence of a listener on a remote server you can do so as follows from the operating system command line:

```
$ tnsping <host>:<port>/<service_name>
```

To further this example, suppose I have a remote client that wants to verify if there's a listener running remotely on the *shrek* server. To verify the listener is up and receiving incoming requests I *tnsping* the *shrek* server on port 1529 via the *TRG* service:

```
$ tnsping shrek:1529/TRG
```

If successful, you'll see output similar to this:

```
Attempting to contact (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=TRG))
(ADDRESS=(PROTOCOL=TCP)
(HOST=shrek)(PORT=1529)))
OK (40 msec)
```

What's confusing is that the prior output only means that the listener is up and receiving requests; it doesn't necessarily indicate that there actually is a service (*TRG* in this example) running on the remote host. For example, look at what happens when I try to connect to the remote service:

```
$ sqlplus system/foo@shrek:1529/TRG
```

This appears in the output:

```
ORA-12514: TNS:listener does not currently know of service requested in
connect descriptor
```

In this situation you should verify which services are registered with the remote listener. Logging on to the remote box, run the following code:

```
$ lsnrctl services
```

The output verifies the listener doesn't have any services registered:

```
The listener supports no services
The command completed successfully
```

In this scenario ensure that the listener has registered the service that you're attempting to remotely contact. You can do this by following the instructions laid out in previous sections that show how to either dynamically or statically register services with the listener.

Listener

The listener is the process that resides on the database server and listens for remote clients attempting to connect to the database. When a remote client attempts to connect to a database it specifies the host name, port, and service associated with the database that it wants to connect to. The listener is configured to listen on one or more ports for services that have been registered with the listener either dynamically or statically.

Starting a Listener

The *lsnrctl* utility allows you start and stop the listener. The *START* option will start the listener process:

```
$ lsnrctl start
```

If nothing has been configured in the *listener.ora* file, you'll see:

```
Connecting to (ADDRESS=(PROTOCOL=tcp)(HOST=shrek)(PORT=1521))
The listener supports no services
The command completed successfully
```

By default the listener will listen for incoming connections on port 1521. Additionally, the default name of the listener is *LISTENER*. You can check the status of the listener with the *STATUS* option:

```
$ lsnrctl status
```

Here's a portion of the output:

```
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Solaris: Version 12.1.0.2.0 -
                          Production
Start Date                25-JAN-2015 18:04:16
Uptime                    0 days 0 hr. 4 min. 58 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /orahome/app/oracle/product/12.1.0.2/db_1/network/
                          admin/listener.ora
Listener Log File         /orahome/app/oracle/diag/tnslsnr/shrek/listener/
                          alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=shrek)(PORT=1521)))
...
```

The listener running by itself with no services registered is of little use. Services must be registered with the listener before the database will accept remote connections. If the listener is listening on port 1521 then by default the *LREG* process periodically checks to see if there are any databases running a box and whether there are any services associated with the database. The *LREG* process performs dynamic service registration.

Service registration makes the listener aware of details about the database running on the box, such as the following:

- Instance name
- Services running within the instance
- Service handlers (dedicated or multiple dispatchers)
- Dispatcher information

If the listener has been configured to listen on a port other than 1521, then you can still have *LREG* perform dynamic registration by setting the *LOCAL_LISTENER* initialization parameter to contain details regarding the listener, such as the host and the port it is listening on.

Modifying the Behavior of the Listener

The most common ways that you'll use to modify the behavior of the listener are setting the listener to listen on a non-default port and naming the listener.

Setting the port

Using the default port of 1521 is almost never recommended. This port number is well known to be the default and is the first port that a hacker will attempt using to remotely access an Oracle database. To set the port to a different value, first stop the listener if it is running:

```
$ lsnrctl stop
```

To configure the port number, navigate to the *ORACLE_HOME/network/admin* directory and add an entry in the *listener.ora* file:

```
LISTENER =
  (DESCRIPTION_LIST=
    (DESCRIPTION=
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1529))
       )
     )
   )
```

These lines of code instruct the listener to listen on the port of 1529 on the host of *shrek*. You can instruct a single listener to listen on multiple ports by adding more addresses to the address list. For example:

```
LISTENER =
  (DESCRIPTION_LIST=
    (DESCRIPTION=
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1529))
        (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1530))
       )
     )
   )
```

Naming the listener

By default when you start the listener process without configuring anything, the default name that Oracle assigns to the listener is *LISTENER*. You can verify the name of a listener that is currently running via the operating system utility *ps*. For example:

```
$ ps -ef | grep tns | grep -v grep
```

In the output the name appears as the text *LISTENER*:

```
oracle   12320     1  0 09:50 ?        00:00:00 /u01/app/oracle/product/
12.1.0.2/db_1/bin/tnslsnr
LISTENER -inherit
```

■ **Tip** If you're in a Windows environment you can check to see whether there's a listener Windows service (don't confuse a Windows service with an Oracle database service) running by right clicking on the task manager bar, starting the task manager, and then viewing services. You can view from the Windows services manager if there's an Oracle listener running.

You can also verify the name of the listener by viewing the output of the *lsnrctl* utility executed with the *STATUS* option. For example:

```
$ lsnrctl status
```

In the output the alias name appears as *LISTENER*:

```
STATUS of the LISTENER
----------------------
Alias         LISTENER
```

If you require more than one listener running on a host then it's common to start different listeners with different names listening on different ports. The following lines of code are placed in the *listener.ora* file and give the listener the name of *DGLIST*:

```
DGLIST =
  (DESCRIPTION_LIST=
    (DESCRIPTION=
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = shrek)(PORT = 1531))
       )
     )
   )
```

To start this listener you need to specify its name (if not using the default name) when executing the *lsnrctl* utility. For instance:

```
$ lsnrctl start dglist
```

If you want to statically register services with this named listener then you must add an additional section to the *listener.ora* file, as follows:

```
SID_LIST_DGLIST =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = DG1) # service name
      (ORACLE_HOME = /u01/app/oracle/product/12.1.0.2/db_1)
      (SID_NAME = TRG) # instance name
     )
  )
```

You can only have one listener listening on the same port on the same host.

Getting help

To view help information for the listener, execute the *lsnrctl* utility with the *help* option:

```
$ lsnrctl help
```

Here's a partial listing of the output:

```
The following operations are available
An asterisk (*) denotes a modifier or extended command:

start           stop            status          services
version         reload          save_config     trace
spawn           quit            exit            set*
show*
```

Now if you want to view more information about a particular command, you can run the *lsnrctl* utility and name the option you need help with:

```
$ lsnrctl help start
```

Summary

This chapter provides you with the basic information you will need to successfully work with simple single instance non-RAC Oracle Net environments. This knowledge lays the foundation for understanding and implementing Oracle Net. With this information you'll be able to more effectively implement and troubleshoot Oracle Net configurations. Elementary concepts such as defining terms, creating and registering a service with the listener, dynamic and static registration, and listener usage were covered.

This chapter was included in this book because in many duplication scenarios you'll be required to use Oracle Net to connect to the target database, auxiliary database, and/or a recovery catalog. You should now be able to quickly configure and successfully use Oracle Net for your basic duplication requirements.

Index

D

E

F, G

H

I, J, K

L

M

N

O

P, Q

R

S

T, U, V, W, X, Y, Z

Oracle RMAN Database Duplication

Darl Kuhn

IOUG independent oracle users group

Apress®

Oracle RMAN Database Duplication

ISBN-13 (pbk): 978-1-4842-1113-7

ISBN-13 (electronic): 978-1-4842-1112-0

Managing Director: Welmoed Spahr
Lead Editor: Jonathan Gennick
Technical Reviewers: Bill Padfield and Fuad Arshad
Editorial Board: Steve Anglin, Mark Beckner, Gary Cornell, Louise Corrigan, Jim DeWolf, Jonathan Gennick, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing, Matt Wade, Steve Weiss
Coordinating Editor: Jill Balzano
Copy Editor: April Rondeau
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

To the readers.
Keep contributing and sharing your knowledge.

Contents

About the Author

Darl Kuhn I'm a DBA/developer working for Oracle. I teach Oracle classes at Regis University in Denver, Colorado, and am also an active member of the Rocky Mountain Oracle Users Group. I enjoy sharing knowledge, and that has led to several book projects over the years.

About the Technical Reviewers

Bill Padfield is an Oracle Certified Professional, working for a large telecommunications company in Denver, Colorado, as a senior database administrator. Bill helps administer and manage a large data warehouse environment consisting of more than 100 databases. Bill has been an Oracle Database administrator for more than 16 years and has been in the IT industry since 1985. Bill also teaches graduate database courses at Regis University and currently resides in Aurora, Colorado, with his wife, Oyuna, and son, Evan.

Fuad Arshad is a senior database architect who has worked with Oracle Database technologies for more than 16 years. He has experience in all aspects of Oracle Database, from management to tuning, and he is an Oracle Certified Expert. He frequently blogs about Oracle at `http://www.fuadarshad.com`. Fuad participates in online forums and social media. He is an active Twitter user, and you can find him there at `http://www.twitter.com/fuadar`. Fuad has presented at conferences, such as Collaborate and Oracle OpenWorld, on topics ranging from Oracle Real Application Clusters to Oracle Database Appliance. Fuad currently works for Oracle Corporation in its North American Sales organization. He is husband to Saba and father to Areej and Ammaar, with whom he tries to spend all of his non-Oracle-related time.

Acknowledgments

Thanks to Jonathan Gennick, Jill Balzano, and the entire Apress staff; it takes a good (coordinated) team to produce a quality book.

Also thanks to many developers and DBAs that I've learned from over the years, including Dave Jennings, Scott Schulze, Bob Suehrstedt, Venkatesh Ranganathan, Valerie Eipper, Mike Tanaka, Simon Ip, Nitin Mittal, Mohan Shanmugavelu, Ric Ambridge, Kamal Chamakura, Dallas Powell, Krishna (KP) Tallapaneni, Laurie Bourgeois, Todd Sherman, Radha Ponnapalli, Mohan Koneru, Kevin O'Grady, Peter Schow, Sujit Pattnaik, Roger Murphy, Barb Sannwald, Pete Mullineaux, Janet Bacon, Shawn Heisdorffer, Mehran Sowdaey, Patrick David, Carson Vowles, Aaron Isom, Tim Gorman, Bill Wiley, Liz Brill, John Biernacki, Joe Stella, Mike Eason, and Jim Stark.