ISBN 1-931971-41-2

13 Ben Rockwood The Sysadmin's Guide to Oracle

13 *Short Topics in* Systems Administration

Rik Farrow, Series Editor

The Sysadmin's Guide to Oracle

Ben Rockwood

13 *Short Topics in*
System Administration

Rik Farrow, Series Editor

The Sysadmin's Guide to Oracle

Ben Rockwood

Published by the USENIX Association
2005

This booklet is dedicated to the memory of my hero, Bob Larson,
who gave more to the UNIX community than most will ever know.

ISBN 1-931971-41-2

To purchase additional copies and for membership information, contact:

The USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA USA 94710
orders@usenix.org
http://www.usenix.org/

First Printing 2005

Contents

Foreword

The Sysadmin's Guide to Oracle, like many books, went through a long, transformative process before it was printed. The result is a short book that covers what a person who may support Oracle 10g as a sysadmin needs to know.

Ben Rockwood set about this project in a way that I found familiar. He set up his own Oracle servers and experimented, based on his research, until he found techniques that worked. This brand of work and writing is really core to much of the UNIX sysadmin experience, where hands-on is the standard for learning.

If you find yourself charged with managing systems that run Oracle, even if you have formerly worked with Postgres or MySQL, this book will be of service to you.

Rik Farrow

Short Topics Series Editor

Introduction

The Oracle Relational Database Management System (RDBMS) has become one of the most powerful and flexible databases available. With ever-expanding functionality provided by options such as Data Guard replication and Real Application Clusters (RAC), it has increasingly blurred the divide between database administrators and system administrators. Nowadays it's becoming essential that sysadmins have a solid understanding of Oracle in order to interface effectively with database admins and architects.

This booklet won't teach you everything about Oracle, and it's not intended to. Instead, you're going to get an overview of the major components of Oracle, specifically Oracle10g. Even though you might never export an Oracle database, it's essential that you understand what it is, so that when your database admins do it, you'll know what they are doing and how to provide for their needs. Throughout the booklet you'll find URLs to Oracle documentation which will provide you with more information on a given topic.

By the end of this booklet you'll have the confidence to log in to Oracle, configure basic networking, manipulate user access, understand the programming interfaces, utilize major tools for importing and exporting data, and interface with the backup system—and, most important, you'll have enough knowledge to learn more about Oracle on your own.

The Relational Model

In this day and age it's easy to take the "simplicity" of the modern relational database for granted. If when you first learned about relational databases you said to yourself, "This is just a bunch of spreadsheets!" you'd have been more or less right. What makes a relational database special is how relationships between parts of the data are managed. In 1985 Dr. E.F. Codd, an IBM researcher, published a list of rules known as "Codd's 12 Rules" that defined how a true RDBMS should be evaluated. Understanding these rules will greatly improve your ability to understand RDBMSes in general and Oracle in particular:

1. Information is represented logically in tables.
2. Data must be logically accessible by table, primary key, and column.
3. Null values must be uniformly treated as "missing information," not as empty strings, blanks, or zeros.

4. Metadata (data about the database) must be stored in the database just as regular data is.
5. A single language (usually SQL) must be able to define data, views, integrity constraints, authorization, transactions, and data manipulation.
6. Views must show the updates of their base tables and vice versa.
7. A single operation must be able to retrieve, insert, update, or delete data.
8. Batch and end-user operations are logically separate from physical storage and access methods.
9. Batch and end-user operations can change the database schema without having to recreate it or the applications built upon it.
10. Integrity constraints must be available and stored in the relational database metadata, not in an application program.
11. The data manipulation language of the relational system should not care where or how the physical data is centralized or distributed.
12. Any row processing done in the system must obey the same integrity rules and constraints that set-processing operations do.[1]

Of these rules the most difficult one to come to terms with is rule 4: this is where *system tables* or *data dictionaries* come in and why it's so difficult to start exploring the database without a basic understanding of the relational database design rules. It's not terribly different from building a Web server that must be configured using a CGI; while it isn't intuitive, it is an admirable design.

You by no means need to bother memorizing this list of rules, but if they're stashed in your long-term memory, retrieval may help you to understand some design decisions used by modern relational databases that might otherwise seem odd.

Play Along at Home

All of the examples and code used throughout this booklet were done using Oracle10g Enterprise Edition on Solaris 10 (Sun Ultra2 Dual UltraSPARC II workstation, 512MB memory) and Gentoo Linux (AMD AthlonMP Dual 1.2Ghz, 1GB memory, Kernel 2.6.8.1). The Oracle Database is available as a "free" download from Oracle.com. Oracle provides you with a 30-day trial period by which to test and explore the product. Please note that although the software will not stop you from running Oracle longer than that, it is illegal to do so.

1. I was unable to acquire a copy of Codd's *The Relational Model for Database Management, 2nd ed.*; these rules are quoted from Kevin Kline and Daniel Kline, *SQL in a Nutshell* (O'Reilly, 2004).

1. Installation and the OFA

Installation of Oracle is pretty simple, although the intertwining of installation and database creation can cause confusion. In fact, it is probably best *not* to create a database during installation. Leave that to the Database Configuration Assistant (DBCA) after Oracle has been installed.

The layout of files in an Oracle installation can be confusing. Often you'll see installations of Oracle in /u01 and data placed in /u02, /u03, and so on. If you do a fresh install of Oracle, though, that installation layout may not be suggested. Why, then, do so many database admins do it that way?

The answer is the OFA (Optimal Flexible Architecture—note: "Optimal," not "Oracle"). The OFA was created by Cary Milsap at the Oracle Users Conference in 1991. The OFA was an attempt to create a standardized set of conventions for Oracle file locations and file naming. The OFA has been widely adopted and is considered an Oracle Best Practice. Besides creating order from chaos, the OFA ensures that your installation will always be able to scale to larger databases, new versions of Oracle, etc.

Installation guidelines combining the OFA and the Oracle installation recommendations are detailed in the following sections.

UNIX Users and Groups

Three UNIX groups should be added: *dba*, *oper*, and *oinstall*. The dba group is used for database install and has SYSDBA database admin privileges. The oper group is optional and maps to the SYSOPER privileges in the database. The oinstall must be created when you install the first time and is the owner of the Oracle Inventory, which is a catalog of installed Oracle software on the system.

Only one user needs to be created: *oracle*. The *oracle* user owns all the Oracle software, and its primary group must be oinstall, with dba and oper as secondary groups. After Oracle is installed, many people change the primary group to dba, but it's not recommended.

Environmental Variables

Several Oracle environmental variables are utilized by various Oracle tools. Essential variables include `ORACLE_SID`, which specifies the Oracle System Identifier (SID) that identifies the database instance you want to act on, and `ORACLE_HOME`, which specifies the location of the Oracle installation. Most of these variables should be defined in the user's .profile. For example:

```
umask 022
ORACLE_BASE=/u01/app/oracle
ORACLE_SID=testdb
ORACLE_HOME=$ORACLE_BASE/product/10.1.0/db_1
ORACLE_PATH=/u01/app/oracle/product/10.1.0/db_1/bin:.
TNS_ADMIN=$ORACLE_HOME/network/admin
```

If `ORACLE_SID` and `ORACLE_HOME` aren't defined, the various Oracle tools such as SQL*Plus won't know which Oracle installation to use or which database to access. You can find a full list of all supported environmental variables in the documentation: http://download-west.oracle.com/docs/html/B10812_02/chapter1.htm#sthref23.

File Locations

The OFA suggests installing in top-level directories named something like /u01, /u02, etc. (you can name them anything you want, so long as you use a consistent naming scheme). Before installing Oracle you should create the directory structure for user oracle and change ownership of the directories to oracle:oinstall. Each top-level directory consists of both app and oradata directories (e.g., /u01/app, /u01/oradata), where the app directories contain user directories for each database user and the oradata directories contain Oracle data files. In this way, you're actually installing Oracle into the oracle user directory. Therefore, you should set this directory (/u01/app/oracle) as the home directory for user oracle.

The following are some examples of Oracle directories that conform to the OFA:

```
/u01/app/oracle/                      Base
/u01/app/oracle/oraInventory          Oracle Inventory
/u01/app/oracle/product/9.2.0         Oracle9i Home
/u01/app/oracle/product/10.1.0/db_1   Oracle10g Home
/u01/app/oracle/admin/                Administrative
/u01/app/oracle/admin/db_name1/       Admin Subtree for a SID
/u01/app/oracle/doc/                  Online Docs
/u01/app/edm/                         Oracle Home Dir for user edm
/u02/oradata/db_name1/                Data for SID
```

When you install Oracle, the oratab file will be installed in /var/opt/oracle. This file lists the available database instances and whether they are startable. If they are defined in oratab as startable, you can start databases using the dbstart script. (Think of dbstart being to Oracle what the startx script is to X11.)

Memory Parameters

Thanks to the massive memory and IPC requirements of Oracle, we need to increase the default memory and IPC-related tunable parameters of the host OS for Oracle to run. If you do not increase these values before you install, the Oracle Installer will require you to adjust them.

The following are the Oracle Installer required minimum values for Solaris, to be set in /etc/system:

```
set noexec_user_stack=1
*
set semsys:seminfo_semmni=100
set semsys:seminfo_semmns=1024
set semsys:seminfo_semmsl=256
set semsys:seminfo_semvmx=32767
*
set shmsys:shminfo_shmmax=4294967295
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=100
set shmsys:shminfo_shmseg=10
```

These are tunable by definition and therefore can be tweaked upward for your environment, but it is not recommended until you have become very familiar with Oracle's behavior.

Please note that some of the above tunables have been deprecated in Solaris 9 and Solaris 10, but the installer will complain if they aren't present. You can either add them to /etc/system, even though they do nothing, or ignore the error generated by the installer when it checks. See the *Solaris Tunable Parameters Reference Manual* at http://docs.sun.com for a complete listing of Solaris tunables for your version.

Troubleshooting Logs

Oracle reports errors (called "exceptions") in *trace files* and *alert logs*. The alert logs are found in `$ORACLE_HOME`/rdbms/log and are database wide. The trace files are per instance and found in `$ORACLE_BASE`/admin/SID/bdump/.

Steps for Installation

We can take all this and set up a basic order for installation:

1. Modify the system tunable for your platform (/etc/system on Solaris) and reboot to take effect.
2. Create the directories /u01/app/oracle and /u01/oradata.
3. Create the groups dba, oinstall, and oper.
4. Create user oracle with primary group oinstall and /u01/app/oracle as the home directory.
5. Add user oracle to the dba and oper groups.
6. Start the Oracle Universal Installer from the Oracle install media.
7. Use /u01/app/oracle/oraInventory as the path for the inventory and /u01/app/oracle/product as the base path for the installation, skipping database creation.
8. After the installer is complete, use DBCA (Database Configuration Assistant) to create your databases.

When you create your databases with DBCA, continue to use the OFA by installing your data files in /u01/oradata/SID or, if you prefer to use a different mount point,

/u02/oradata/SID. Obviously, you can get creative with your data layout as it maps to your disk subsystem. Some database admins will request a local mount for /u01 to install Oracle and then two or more /u0*n* directories with different specs for data files. In this way we can distribute files onto disks with different access patterns or tuning specifications.

Please note that some operating system versions, such as Solaris 10, are not supported by Oracle10g. They will run the database without problems, but the installer will report that you are using an unknown or unsupported operating system. To forcibly install Oracle, you can start the installer with the "-ignoreSysPrereqs" argument to runInstaller.

2. Oracle Basics

First we'll cover how to do the most basic operations:

1. Create a database.
2. Connect to and start the database.
3. Create a table.
4. Add data to the table.
5. Create an index.
6. Query the table.
7. Create and query a view.
8. Destroy objects.
9. Stop the database.
10. Destroy the database.

Create a Database

Two methods are available for creating a new database: the Database Configuration Assistant (DBCA) GUI and the `CREATE DATABASE` SQL statement. For simplicity we'll use the DBCA. You can read about creating a database using the SQL interface in Chapter 2 of the *Oracle Database Administrator's Guide*.

To use the configuration assistant, export your display (if you're not on the console) and run DBCA:

```
$ export DISPLAY=10.10.1.100:0
$ pwd
/u01/app/oracle/product/10.1.0/db_1/bin
$ ./dbca
```

We'll then use the following steps in the GUI. The step numbers noted (e.g., "In step 1") refer to the numbered steps that DBCA uses to identify your progress.

1. In step 1, choose "Create a database."
2. In step 2, choose "General Purpose."
3. In step 3, we'll name the database test.cuddletech.com with the SID *testdb.*
4. In step 4, we'll leave "Configure the Database with Enterprise Manager" checked and also check "Enable Email Notifications," specifying both our local SMTP server and our email address.
5. In step 5, we'll use the same password for all accounts. Here we'll use "passwd".

6. In step 6, we'll use regular File System storage.
7. In step 7, we'll choose "Use Common Location for All Database Files"; following the OFA, we'll use /u02/oradata as the location.
8. In step 8, we'll disable flash recovery by unchecking all boxes.
9. In step 9, we'll uncheck "Sample Schemas."
10. In step 10, we'll use the default settings for memory management. In my case that's 120MB for the shared pool, 24MB for the buffer cache, 8MB for the large pool, and 24MB for the PGA. The only thing to change is "Java Pool": set it to 0.
11. In step 11, look over the layout of the storage. Nothing needs to be changed here.
12. At last, in step 12 we can actually create the database or save it as a template. Make sure "Create Database" is checked and click "Finish."
13. We'll now have the chance to look at our configuration and save an HTML copy of it. When done, create the database.
14. The database will now build. On my Sun Blade 100 this process took about 10 minutes.
15. Finally, you'll get a confirmation dialog with the database name, SID, Server Parameter Filename, and an Enterprise Manager URL.

After the database has been created, you can go examine the files it created:

```
# cd /u02/oradata/testdb
# ls -alh
total 1451556
drwxr-xr-x 2 oracle     oinstall    512 Oct 5 16:43 .
drwxr-xr-x 4 oracle     dba          512 Oct 5 16:40 ..
-rw-r——-   1 oracle     oinstall    2.7M Oct 5 16:53 control01.ctl
-rw-r——-   1 oracle     oinstall    2.7M Oct 5 16:53 control02.ctl
-rw-r——-   1 oracle     oinstall    2.7M Oct 5 16:53 control03.ctl
-rw-r——-   1 oracle     oinstall    10M Oct 5 16:53 redo01.log
-rw-r——-   1 oracle     oinstall    10M Oct 5 16:46 redo02.log
-rw-r——-   1 oracle     oinstall    10M Oct 5 16:51 redo03.log
-rw-r——-   1 oracle     oinstall    210M Oct 5 16:53 sysaux01.dbf
-rw-r——-   1 oracle     oinstall    430M Oct 5 16:53 system01.dbf
-rw-r——-   1 oracle     oinstall    20M Oct 5 16:43 temp01.dbf
-rw-r——-   1 oracle     oinstall    25M Oct 5 16:53 undotbs01.dbf
-rw-r——-   1 oracle     oinstall    5.0M Oct 5 16:46 users01.dbf
```

There will also be files in your Oracle admin directory (OFA path: /u01/app/oracle/admin/[SID]) and the dbs directory (OFA path: /u01/app/oracle/product/10.1.0/db_1/dbs).

If you want to play with Enterprise Manager, you can do so now. Go to the URL listed in the final dialog of the DBCA session. At the login screen use the username "sys", the password "passwd" (as we specified during creation) and from the drop-down menu choose "SYSDBA". After you log in and agree to the licensing information

terms, you'll get the full Enterprise Manager experience. You'll see that the database has already been started and is up. On the "Administration" tab you can really play with some nifty things: You can create nearly any database component here without ever touching SQL*Plus.

If you are playing with the Enterprise Manager, go ahead and shut down the database so that we can see how to start it up using SQL*Plus.

Connect to and Start the Database

Database startup can be done using either SQL*Plus or the Enterprise Manager (EM). We'll use SQL*Plus. There are a variety of ways to log in to SQL*Plus. Using the "/nolog" argument followed by a connect string is best, because specifying the username and password on the command line make it visible to system tools such as ps or pargs.

```
$ export ORACLE_SID=testdb
$ sqlplus /nolog
. . .
SQL> CONNECT sys/passwd AS SYSDBA
Connected to an idle instance.
SQL> STARTUP
ORACLE instance started.
Total System Global Area          184549376 bytes
Fixed Size              1300928 bytes
Variable Size           157820480 bytes
Database Buffers        25165824 bytes
Redo Buffers            262144 bytes
Database mounted.
Database opened.
SQL> quit
Disconnected from Oracle Database 10g
$ ps -ef | grep -i testdb
oracle     1624         1           0       17:16:20       0:00
ora_qmnc_testdb
oracle     1612         1           1       17:16:08 ?
0:01ora_cjq0_testdb
. . .
```

Many additional startup arguments can be used: `STARTUP NOMOUNT`, for example, to keep the database from mounting, or `STARTUP FORCE`, to force it to start.

Once startup completes, the database should be open, mounted, and in read/write mode.

Create a Table

If you disconnected from SQL*Plus, reconnect and we'll try creating a table:

```
SQL> create table address_book (
 2 id number,
 3 name varchar2(30),
 4 home_num varchar2(12),
 5 cell_num varchar2(12),
 6 location varchar(40)
 7 );
Table created.
```

Add Data to the Table

Let's add some data to the table:

```
SQL> insert into address_book
 2 values (1, 'Ben Rockwood', '510-555-1234', '650-555-2345',
'Menlo Park, CA');
1 row created.
SQL> insert into address_book
 2 values (2, 'Tamarah Rockwood', '510-555-4443', NULL, 'Menlo
Park, CA');
1 row created.
SQL>
```

Create an Index

Index creation works just as you'd expect:

```
SQL> create index home_num_idx
 2 on address_book (id, home_num);
Index created.
```

Query the Table

Grabbing data from the table is trivial using standard SQL:

```
SQL> select name from address_book;
NAME
-----------------
Ben Rockwood
Tamarah Rockwood
```

Create and Query a View

Views make tables a little more comfortable to use as they grow larger:

```
SQL> create view cell_view as
 2 select name, cell_num
 3 from address_book;
View created.
SQL> select * from cell_view;
```

```
NAME                CELL_NUM
------------------------------
Ben Rockwood        650-555-2345
Tamarah Rockwood
```

Destroy Objects

Using DROP statements, we can easily destroy objects that are no longer needed:

```
SQL> drop view cell_view;
View dropped.
SQL> select * from address_book;
ID  NAME              HOME_NUM         CELL_NUM
-------------------------------------------------
LOCATION
-------------------------------------------------
1   Ben Rockwood      510-555-1234     650-555-2345
Menlo Park, CA
2   Tamarah Rockwood  510-555-4443
Menlo Park, CA
SQL> drop table address_book;
Table dropped.
```

Stop the Database

If you're not already logged in, log in now and shut down the database:

```
bash-2.05$ sqlplus /nolog
. . .
SQL> connect sys/passwd as sysdba;
Connected.
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> quit
```

There are a variety of shutdown methods. Using shutdown without an argument is rare, because the database won't shut down until all connections have disconnected willingly, during which time it won't allow any new connections. `shutdown immediate` will safely close out all current connections regardless of whether or not they want to.

Destroy the Database

Databases can be deleted like any other object, using the DROP SQL statement. In order to drop the database it must be mounted and closed, mounted exclusively (not in shared mode), and flagged as RESTRICTED. The easiest way to do this is by using EM (but then, most things are). Once your database is in the proper state, issuing the SQL `drop testdb` will destroy the test database we created.

3. Poking Around Inside Oracle

What separates sysadmins from everyone else? We want to understand everything. Being highly analytical people by nature, we need to understand not only how to do something but the environment in which we operate. But, sadly, when it comes to Oracle that's a tall order, especially without the assistance of the Enterprise Manager or tools like TOra.

I find smaller databases such as PostgreSQL and MySQL very refreshing, because they are so easy to dig around in. In PostgreSQL, if you want to see all the tables, just issue a `\dt` and look at the list. Want to see the indexes? Type `\di`, and there they are. This makes it easy to poke around when you are either new or lost in your environment. With Oracle it's a little harder. When I really started playing with Oracle I was frustrated by the *blindness* of the whole experience. I could only see as far as the tip of my nose. But I soon realized just why you can't see everything with such clarity in Oracle—there is just too much!

Oracle keeps so many internal tables that it's hard to browse around in them. If you pull up Enterprise Manager, you'll see (sit down for this) 1,468 tables and 3,470 views. This immediately answers the question, "Why doesn't SQL*Plus have a `\dt` feature like PostGreSQL?" Thankfully, you can find lists of the commonly used tables, such as http://www.techonthenet.com/oracle/sys_tables/.

The `ALL_*` tables are especially interesting. For instance, using the SQL `COUNT()` function, we can count the number of tables listed in the `ALL_TABLES` system table:

```
SQL> select COUNT(*) from ALL_TABLES;
 COUNT(*)
------
1468
```

When managing Oracle you're going to need to query the system tables a lot, so keeping your SQL skills sharp and a reference or cheat sheet nearby is a good idea. Here's a common example:

```
SQL> select * from dba_users
  2 where username = upper('benr');
USERNAME                  USER_ID                PASSWORD
------------------------------------------------------------
ACCOUNT_STATUS            LOCK_DATE              EXPIRY_DA
------------------------------------------------------------
```

```
DEFAULT_TABLESPACE         TEMPORARY_TABLESPACE  CREATED
-------------------------------------------------------------
PROFILE                    INITIAL_RSRC_CONSUMER_GROUP
-------------------------------------------------------------
EXTERNAL_NAME
-------------------------------------------------------------
BENR                       58                    BEFEAB8A2CDD5A85
OPEN
USERS                      TEMP                  08-OCT-04
```

Looking at all that output, it's hard to see what's happening. For this reason, whenever possible limit your queries to just what you really need. You'll also notice that when you're searching (using the "where" clause), the search parameter is case-sensitive, which can be confusing because so much of SQL is case-insensitive. Almost everything is stored in uppercase, so you can either search for "BENR" or you can use the SQL function `upper()` to convert to uppercase.

Let's try outputing that table again, with a little less cruft this time:

```
SQL> Select username, user_id, profile from dba_users
 2 where username = 'BENR';
USERNAME          USER_ID             PROFILE
-------------------------------------------------------------
BENR              58                  DEFAULT
```

Much better.

The Data Dictionary

The *data dictionary* is the set of tables and views that store Oracle's own data. The tables in the data dictionary are broken down into three sets of views: `ALL_`, `DBA_`, and `USER_`. The `ALL_` views display all the information accessible to the current user. The `DBA_` views display all relevant information in the entire database and are only intended for admins. Finally, the `USER_` views display all the information from the schema of the current user. The difference between the `USER_` and `ALL_` views is that you can only look around in your own schema using `USER_` views, but in the `ALL_` views you see all the schemas you have permission to view. Because these are views, most of the data is duplicated between them: for instance, `USERS_USERS`, `ALL_USERS`, and `DBA_USERS`.

You can find a complete listing of all the data dictionary tables and views in the *Oracle Database Reference* manual: http://download-west.oracle.com/docs/cd/B12037_01/server.101/b10755/ toc.htm.

Essential System Tables

What follows is a list of tables that will make your life easier and help you feel less blind and helpless. This is by no means a complete list, just some important tables to get you on your way poking about and learning the lay of the land.

Table 4.1: Some Essential Oracle System Tables

Table	*Description*
ALL_CATALOG	All tables, views, synonyms, sequences accessible to the user
ALL_INDEXES	Descriptions of indexes on tables accessible to the user
ALL_OBJECTS	Objects accessible to the user
ALL_OBJECT_TABLES	Description of all object tables accessible to the user
ALL_TABLES	Description of relational tables accessible to the user
ALL_TRIGGERS	Triggers accessible to the current user
ALL_TYPES	Description of types accessible to the user
ALL_UPDATABLE_COLUMNS	Description of all updatable columns
ALL_USERS	Information about all users of the database
ALL_VIEWS	Description of views accessible to the user
DBA_OBJECTS	All objects in the database
DBA_ROLES	All roles which exist in the database
DBA_ROLE_PRIVS	Privileges granted to users and roles
DBA_SOURCE	Source of all stored objects in the database
DBA_TABLESPACES	Description of all tablespaces
DBA_TAB_PRIVS	All privileges on objects in the database
DBA_TS_QUOTAS	Tablespace quotas for all users
DBA_USERS	Information about all users of the database
DBA_VIEWS	Description of all views in the database
DICTIONARY	Description of data dictionary tables and views
GLOBAL_NAME	Global database name
PRODUCT_COMPONENT_VERSION	Version and status information for component products
SESSION_PRIVS	Privileges which the user currently has set
SESSION_ROLES	Roles which the user currently has enabled
SYSTEM_PRIVILEGE_MAP	Maps privilege type numbers to type names

The Dollar Views

I sometimes hear these mistakenly referred to as the "dollar tables," but they're actually views. These are the system views that are named V$[something]. Because they are in the SYS schema, you'll sometimes see them more explicitly named as SYS.V$[something]. These views are unique because they are dynamic and present in memory, which means they can be queried even when the instance isn't open. If you need to deal with a database that is crippled, these views may be all that's accessible to you. When looking at system views you may also see GV$[something] views; these are for RAC installations where you want to see the Global View.

Views of interest include `V$ACTIVE_SERVICES`, which contains data regarding all active services provided by the instance; `V$CLIENT_STATS`, where you can find data regarding client connections to the instance; `V$CONTROLFILE`, which contains the locations of the control files and their status; and `V$DATABASE`, which contains information about the database from the control files.

You can learn about all of these views in the *Oracle Database Reference* manual: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10755/dynviews_part.htm#i403961.

4. Files and Components

As a sysadmin you need to be able to divide the various files in Oracle into logical groups mentally. Knowing what goes where will help you better understand your role in maintaining Oracle. For this discussion, we'll assume that you have installed Oracle in accordance with the OFA.

We can divide the installation into two main categories: distribution files and database files. Oracle distribution files would be everything that you installed when you installed Oracle. Database files would be all the files that pertain to a given instance of the database.

It's important here to clarify the difference between a *database* and an *instance*. A *database* is the collection of files that store your data and all the metadata and configuration information which govern how that data is accessed, utilized, etc. An *instance* is a group of processes that make your database accessible. For example, the data file users01.dbf is a component of the database and the logwriter process ora_lgwr_SID is a component of the instance.

Database Files

When you create a database, you'll be creating several files:

Control Files	Usually three, named control01.ctl–control03.ctl.
Data Files	The default DBCA configuration is to create five data files: system01.dbf, undotbs01.dbf, sysaux01.dbf, users01.dbf, and temp01.dbf.
Redo Logs	Typically three, named redo01.log–redo03.log.
Admin Files	A series of directories and files usually in $ORACLE_BASE/admin/[SID].

A control file can be thought of as the Oracle equivalent of a DiskSuite state database. It contains references and metadata pertaining to all the database files, such as data files and redo logs. Typically there are three or more copies of the control file, like DiskSuite's state database replicas. This guarantees that if one control file is lost or corrupt, we still have a usable copy. Control files supposedly can be rebuilt if they are lost or destroyed, but it isn't easy. If there is one file you want to make sure you're backing up, the control file is the one.

The data files contain the actual data. These files are often referred to as *tablespaces*. Within the tablespace reside all the tables that make up your database. For example,

the "users tablespace" is in the users01.dbf data file. Temp tables are written to the *temp tablespace,* which is the temp01.dbf data file.

Redo logs are the database equivalent of a filesystem journal. When changes are made to the database, the changes are written to the redo log. If data is lost, we can recover the lost changes by replaying the redo logs just as we'd replay a VERITAS filesystem journal on an inconsistent file system. Redo logging is enabled by default but can be turned off, if you choose, by enabling the NOLOGGING parameter. There are typically three or more redo logs, which are written in a round-robin fashion. Each one fills up until all the logs are full and it begins overwriting the first again.

On a related note, redo logs can be *archived*. When a database is in `ARCHIVELOG` mode the redo logs can be written out to archive logs and stored elsewhere. The upside is that you could replay weeks of transactions back into the database if you wanted to. The downside is that if the so-called archive log destination (the directory where archived redo logs get stored) becomes full or unwritable, the instance will shut down! Typically a cron or backup job regularly backs up or moves the archived redo logs to tape or to a permanent storage location, ensuring that the archive destination never fills up.

Additionally, several administrative directories will be created in $HOME/admin/[SID] for each database:

bdump	Background dump directory (alert logs and trace files)
cdump	Core dump directory
udump	User dump directory
create	Creation logs
PFILE	Parameter dile directory

Dump directories contain trace files, denoted by the suffix *.trc*. These traces are dumped by various Oracle tools when certain actions are carried out or an error condition is encountered. An experienced DBA can use trace files to debug problems with the database and to provide information to Oracle support. The most useful file for the sysadmin is the alert log, which is the Oracle equivalent of a syslog file. In the alert log (typically named alert_[SID].log) we may find warnings or errors that affect the database. Additionally, some database applications and HA cluster agents will actively monitor the alert log to help determine the state of the database.

Initialization Parameters

Several files contain lists of initialization parameters: init.ora files, PFILEs (parameter files), and SPFILEs (server parameter files). Each database instance will have a PFILE in its admin directory. An init.ora will exist in the $ORACLE_HOME/dbs/ directory. In the PFILE directory of each instance's admin directory you'll also find an init.ora.

The parameter files contain a variety of Oracle tunables, including the database block size (`db_block_size`), database cache size (`db_cache_size`), number of open cursors (`open_cursors`), the database name (`db_name`) and domain (`db_domain`),

resource limits (processes), size of all the various pools (e.g., `large_pool_size`), and the location of the control files (`control_files`). The parameters are read by Oracle when you start an instance, as the name *initialization* implies. Once this file is read and processed, the control files will point the instance at the rest of the data files.

SPFILEs are becoming increasingly important. Unlike the init.ora PFILE, the SPFILE shouldn't be hand-edited. Oracle has been making more and more parameters dynamically changeable. In older releases of Oracle, changing parameters would have involved shutting down the database, modifying the parameter in the PFILE, and then restarting the instance. Now many of these parameters can be changed with an `ALTER` statement and take effect immediately, without downtime. The problem used to be that if you changed these dynamic parameters, the changes wouldn't be persistent across database restarts, so the DBA would have to be careful to update the PFILE after making a change. Obviously this became a problem quickly as it was easy to forget to update manually. To combat this problem, dynamically changed parameters were recorded in the SPFILE. However, you needed to explicitly create an SPFILE by converting your existing PFILE (e.g., `CREATE SPFILE 'SPFILESID.ora' FROM PFILE='initSID.ora';`). You can look in the v$parameter system view to see where your SPFILE is. In 10g it seems that an SPFILE is created automatically:

```
SQL> select name,value from v$parameter where name='SPFILE';
NAME
---------------------------------------------------------
VALUE
---------------------------------------------------------
SPFILE
/u01/app/oracle/product/10.1.0/db_1/dbs/spfiletest.ora
```

When databases are created using DBCA, they use an SPFILE by default and are located in $ORACLE_HOME/dbs.

Oracle Startup

The best way to understand how all the various files work together is to examine the Oracle startup process.

When Oracle starts an instance, it reads the SPFILE or PFILE to determine the initialization parameters. It uses these parameters to allocate the System Global Area (SGA) and create background processes. All this is done without associating a database with the instance! At this point the instance is started but not mounted, what some people call "started in nomount mode," because you can reach this state by using the SQL*Plus command `startup nomount`:

```
SQL> startup nomount;
ORACLE instance started.
Total System Global Area       184549376 bytes
Fixed Size                     1300928 bytes
```

```
Variable Size                  157820480 bytes
Database Buffers               25165824 bytes
Redo Buffers                   262144 bytes
SQL> quit
# ps -ef | grep -i ora_
 oracle       720     1      0 14:14:20 ?  0:00 ora_reco_test
 oracle       710     1      0 14:14:19 ?  0:00 ora_mman_test
 oracle       708     1      0 14:14:19 ?  0:00 ora_pmon_test
 oracle       712     1      0 14:14:19 ?  0:00 ora_dbw0_test
 oracle       718     1      0 14:14:19 ?  0:00 ora_smon_test
 oracle       714     1      0 14:14:19 ?  0:00 ora_lgwr_test
 oracle       716     1      0 14:14:19 ?  0:00 ora_ckpt_test
 oracle       726     1      0 14:14:20 ?  0:00 ora_s000_test
 oracle       724     1      0 14:14:20 ?  0:00 ora_d000_test
 oracle       722     1      0 14:14:20 ?  0:00 ora_cjq0_test
```

When a database is mounted, the data files are associated with the instance. As a test, you can rename the data file directory (where the control and data files are) and start up the instance without mounting: You won't get an error, because Oracle isn't interested in anything but the parameter files at this point. Even though the PFILE specifies the location of the control file, it hasn't tried to open the control files yet, so it won't complain.

This last revelation is extremely important. Why? You'll notice that a lot of the documentation, particularly regarding recovery, will tell you to connect to an instance even though it's in need of major recovery. At first glance, in some cases, you'll scratch your head trying to figure out why they expect you to start an instance of a database that's been destroyed. Well, now you know.

In the next phase of database startup, the database is mounted. In this step we associate the control, data, redo, and other database-related files with the running instance. If you are missing files, this is where you'll get your error. If you started your instance using `nomount` you can't use the `startup` command again, but you can use `alter database` statements to change the state of your instance.

Let's look quickly at what happens when you mount your database with the instance already running but with the all the data and control files missing (because we renamed the data directory):

```
SQL> alter database mount;
alter database mount
*
ERROR at line 1:
ORA-00205: error in identifying controlfile, check alert log for
more info
SQL> quit
# tail /u01/app/oracle/admin/test/bdump/alert_test.log
alter database mount
Wed Oct 13 14:28:12 2004
```

```
ORA-00202: controlfile: '/u02/oradata/test/control01.ctl'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
Wed Oct 13 14:28:12 2004
Controlfile identified with block size 0
Wed Oct 13 14:28:12 2004
ORA-205 signalled during: alter database mount...
```

Notice that it complains about the first control file and not the data files. That's because the parameter file has a listing of all the control files, and the control file is responsible for storing information about all the other data files and resources used by the database. If the control file can't be read, the database doesn't know what else should exist! This is why you must be careful to keep good system backups of your control files. This is also why Oracle maintains multiple copies (typically three) of the control file for safety.

Let's put all the data files back and try mounting the database again.

```
# mv test.HOLD/ test
# sqlplus sys/passwd as sysdba
. . .
SQL> alter database mount;
Database altered.
```

Looking at the processes on the system, you'll notice after mounting the database that no change has occurred to the processes.

Once an instance is started using a PFILE and the mount process has used the control file(s) to associate the data files with the instance, we need to *open* the database. Until a database is opened it is not accessible: It's equivalent to starting a system in single-user mode. Some amount of interaction with the database is available at this stage, but it's limited to *fixed tables* and *views*. The fixed tables and views are those in the data dictionary (Oracle's internal configuration tables) and are preceded with v$:

```
SQL> select status from v$instance;
STATUS
--------
MOUNTED
```

To open the database for normal access, we can alter the database again:

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

The shutdown process is simply the opposite of the startup:

```
SQL> shutdown immediate;
Database closed.
```

```
Database dismounted.
ORACLE instance shut down.
```

Oracle Distribution Files

The Oracle distribution is all the stuff written to disk during the install process (assuming we don't create a database during the install). During install we'll copy all the Oracle binaries into our $ORACLE_HOME directory, which according to the OFA is /u01/app/oracle/product/10.1.0/db_1 for Oracle10g. In here you've got all the binaries that you'll need to work with Oracle.

The other set of files created during install is the Oracle Inventory, found according to the OFA in /u01/app/oracle/oraInventory. The Inventory is used by the Oracle Universal Installer to record what products have been installed, where they were installed, and how. The Oracle Inventory doesn't impact databases in any way; it only affects the installer. You can see a list of all the products and options you have installed in the oraInventory/Components directory.

5. Users and Permissions

Except for security, permissions is, I think, the most confusing topic pertaining to Oracle. The line between the UNIX user (oracle) and Oracle users (e.g., sys) is quite blurry. Just getting logged in to SQL*Plus can be a major task when you aren't sure whether it wants UNIX or Oracle usernames and passwords. I'll try to demystify this a bit.

Users and Passwords

Authentication can be done using *password authentication* (also called internal authentication) or *OS authentication* (external authentication).

If the database is configured for OS authentication you can log in to the database based on your UNIX UID, without authenticating to Oracle. OS authentication is enabled or disabled based on the `os_roles` parameter. When you create a database using DBCA, OS authentication is disabled by default; in fact, you won't be able to enable it unless you edit the initialization parameters in DBCA's step 10 by selecting "All Initialization Parameters" and then selecting "Show Advanced Parameters." If you scroll down the long list you'll see that `os_roles` is set to false. You can change it to true if you choose, but the defaults tell you that Oracle isn't keen on you doing so. Therefore: don't.

In step 5 the DBCA will ask you to assign passwords to the default Oracle accounts—SYS, SYSTEM, SYSMAN, and DBSNMP (descriptions are taken directly from the DBCA help):

SYS	Owns all base tables and the user-accessible view of the data dictionary (Oracle configuration information). No Oracle user should ever alter (update, delete, or insert) any rows or schema objects in the sys schema, because such activity can compromise data integrity. The security administrator must keep strict control of this central account.
SYSTEM	Creates additional tables and views that display administrative information, and internal tables and views used by various Oracle options and tools.
SYSMAN	The Enterprise Manager super admin account. This EM admin user can create and modify other EM admin accounts, as well as administering the database instance itself.

DBSNMP Used by EM to monitor the database. EM uses this account to access performance stats about the database. The dbsnmp credentials are sometimes referred to as the monitoring credentials.

A user can also connect with SYSDBA or SYSOPER privileges. When you `connect sys/passwd as sysdba`, you're connecting as the SYS user and requesting SYSDBA privileges. Because the SYS user is the Oracle equivalent of the UNIX root user, Oracle makes you specify the amount of control you have, which is why you'll get an error if you try to connect without specifying the privileges:

```
SQL> connect sys/passwd
ERROR:
ORA-28009: connection to sys should be as sysdba or sysoper
SQL> connect sys/passwd as sysdba
Connected.
```

SYSDBA and SYSOPER privileges differ in that SYSDBA can do anything (just like root), while SYSOPER allows everything except the ability to look at user data. Both sets of privileges allow you to `alter database`, `create spfile`, `startup` and `shutdown`, and `alter database archivelog`, and they include RESTRICTED SESSION privileges. However, only SYSDBA can `create database` or `drop database`, and the `alter database recover` options for SYSOPER are limited to complete recovery only.

Adding New Users

If you really poke around in Oracle you'll find that the sys account is blocked from a variety of tasks, so we need to create some users. The easiest way to add users is with the Enterprise Manager. In case you have to use SQL, here's how:

```
bash-2.05$ sqlplus /nolog
. . .
SQL> connect sys/passwd as sysdba;
Connected.
SQL> create user ben profile default identified by passwd
 2 default tablespace users
 3 temporary tablespace temp
 4 account unlock;
User created.
SQL> grant connect to ben;
Grant succeeded.
SQL> alter user ben quota unlimited on users;
User altered.
```

Here, as SYSDBA, we've created the user "ben" with the default profile identified by the password "passwd". The user's default tablespace is "users" and temporary tablespace is "temp", and the account is unlocked. In the second statement we grant the "connect" role to user ben. And in the third statement we alter ben's quota on the users tablespace.

Now let's log in as ben:

```
bash-2.05$ sqlplus /nolog
. . .
SQL> connect ben/passwd
Connected.
SQL> create table mytable (
 2 id number(2),
 3 name varchar2(30)
 4 );
create table mytable (
*
ERROR at line 1:
ORA-01950: no privileges on tablespace 'USERS'
```

The user has no privileges because the user wasn't defined with any *roles*. Each user must be assigned one or more security roles. Note that even though our second SQL statement granted the CONNECT role to the user, that wasn't enough. Let's spend a minute sorting out the differences among privileges, roles, and profiles.

Privileges, Roles, and Profiles

Let's define them first:

Privileges	A user privilege is a right to execute a particular type of SQL statement, or a right to access another user's object, execute a PL/SQL package, and so on. The types of privileges are defined by Oracle.
Roles	Roles are created by users (usually administrators) to group together privileges or other roles. They are a means of facilitating the granting of multiple privileges or roles to users.
Profiles	Profiles define resource limits imposed upon a user account. The "default" profile sets all resource limits to "unlimited."

So we can assert a good amount of control over the user here by bundling privileges into roles, then granting those roles to user accounts, and then further controlling the resource usage of the account with a profile. In almost all cases the default profile will be used.

Let's look at roles in more depth. Here are some of the predefined roles Oracle makes available to us:

CONNECT	Includes the following system privileges: alter session, create cluster, create database link, create sequence, create session, create synonym, create table, create view.
RESOURCE	Includes the following system privileges: create cluster, create indextype, create operator, create procedure, create sequence, create table, create trigger, create type.
DBA	All system privileges with admin option.

EXP_FULL_DATABASE Provides the privileges required to perform full and incremental database exports: select any table, backup any table, execute any procedure, execute any type, administer resource manager, and insert, delete, and update on the tables SYS.INCVID, SYS.INCFIL, and SYS.INCEXP. It also includes EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE.

IMP_FULL_DATABASE Provides the privileges required to perform full database imports. Includes an extensive list of system privileges (use view DBA_SYS_PRIVS to view these privileges) and the EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE.

By utilizing the data dictionary (Oracle configuration tables) we can check the current set of roles and privileges. The DBA_ROLES table contains all the roles available, and DBA_ROLE_PRIVS contains the user-to-role mappings. If you're not sure which roles are assigned to a user, check this table:

```
SQL> select * from DBA_ROLES;
ROLE                    PASSWORD
------------------------------
CONNECT                 NO
RESOURCE                NO
DBA                     NO
SELECT_CATALOG_ROLE     NO
EXECUTE_CATALOG_ROLE    NO
...
SQL> select * from DBA_ROLE_PRIVS;
GRANTEE         GRANTED_ROLE    ADM  DEF
--------------------------------------
BEN             CONNECT         NO   YES
DBA             OLAP_DBA        NO   YES
DBA             XDBADMIN        NO   YES
. . .
```

Just as we did when creating the user, we can use SQL's `grant` statement to grant new roles to a user. We can also use the `revoke` statement to remove a role from the user.

```
SQL> grant resource to ben;
Grant succeeded.
SQL> select * from DBA_ROLE_PRIVS where grantee = 'BEN';
GRANTEE         GRANTED_ROLE    ADM  DEF
--------------------------------------
BEN             CONNECT         NO   YES
BEN             RESOURCE        NO   YES
SQL> revoke resource from ben;
Revoke succeeded.
```

```
SQL> select * from DBA_ROLE_PRIVS where grantee = 'BEN';
GRANTEE   GRANTED_ROLE  ADM   DEF
-------------------------------------
BEN       CONNECT       NO    YES
```

Changing Passwords

Passwords can be changed by simply altering the user:

```
SQL> ALTER USER "BENR" IDENTIFIED BY "passwd";
User altered.
```

6. The Listener

The listener allows remote connection to the database. It's part of the larger Oracle Net Services framework. (FYI: In the past Oracle used the marketing terms Net8 and SQL*Net for basically the same thing—see http://www.orafaq.com/faqnet.htm for the few differences.)

There are three interfaces for managing Oracle network configuration: Enterprise Manager, the Oracle Net Configuration Assistant GUI (the binary is *netca*), and the CLI tools (*lsnrctl*, the Listener Control Utility, and *cmctl*, the Oracle Connection Manager Control Utility).

At a high level you need to do two things: configure and start the listener, and then configure name resolution on the clients. Both of these are easily done using netca or EM, but we'll look at the plain files so that you'll be able to do it without these tools.

Configuring the Listener

The listener is easily configured in the $ORACLE_HOME/network/admin/listener.ora configuration file. Here is a basic configuration:

```
LISTENER =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = 10.10.0.130)(PORT = 1521))
  )
 )
SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (ORACLE_HOME = /u01/app/oracle/product/10.1.0/db_1)
   (SID_NAME = test)
  )
 )
```

The first section is labeled "LISTENER"; this is the name of the listener itself, and can be anything you like, although "LISTENER" is the traditional name. Then we have some nested configuration parameters, the most important of which is the address section, which in this case specifies the listener to use TCP, on host address 10.10.0.130 port 1521. Port 1521 is the traditional default port, but again you can use anything you like.

The SID_LIST_LISTENER section defines our essential parameters, namely SID_NAME (same as $ORACLE_SID) and ORACLE_HOME (same as $ORACLE_HOME). The name "SID_LIST_LISTENER" is derived from the LISTENER name, so if you did change the listener's name to "MYLISTENER", for instance, your second section would be "SID_LIST_MYLISTENER".

If you wanted to have multiple databases specified you could nest more SID_DESC parameters in the SID_LIST. In the same way, if you wanted to listen on different IP addresses you could use multiple DESCRIPTION sections in the DESCRIPTION_LIST. For example:

```
LISTENER =
 (DESCRIPTION_LIST =
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = 10.10.0.130)(PORT = 1521))
  )
  (DESCRIPTION =
   (ADDRESS = (PROTOCOL = TCP)(HOST = 10.20.0.10)(PORT = 1522))
  )
 )
SID_LIST_LISTENER =
 (SID_LIST =
  (SID_DESC =
   (ORACLE_HOME = /u01/app/oracle/product/10.1.0/db_1)
   (SID_NAME = test)
  )
  (SID_DESC =
   (ORACLE_HOME = /u01/app/oracle/product/10.1.0/db_1)
   (SID_NAME = anotherdb)
  )
 )
```

For complete details on syntax, please check out the *Oracle Database Net Services Reference Guide*: http://download-west.oracle.com/docs/cd/B12037_01/network.101/b10776/toc.htm.

Starting the Listener

The listener can be started either before or after the database instance is started. Start the listener with a simple `lsnrctl` start:

```
$ lsnrctl start
LSNRCTL for Solaris: Version 10.1.0.2.0 - Production on 11-OCT-2004
15:42:55
Copyright (c) 1991, 2004, Oracle. All rights reserved.
Starting /u01/app/oracle/product/10.1.0/db_1/bin/tnslsnr: please
wait...
TNSLSNR for Solaris: Version 10.1.0.2.0 - Production
System parameter file is /u01/app/oracle/product/10.1.0/db_1/
network/admin/listener.ora
```

```
Log messages written to /u01/app/oracle/product/10.1.0/db_1/
network/
 log/listener.log
Listening on:
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=10.10.0.130)
 (PORT=1521)))
Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=10.10.0.130)
 (PORT=1521)))
STATUS of the LISTENER
------------------------
Alias LISTENER
Version TNSLSNR for Solaris: Version 10.1.0.2.0
Start Date    11-OCT-2004 15:42:55
Uptime 0 days 0 hr. 0 min. 0 sec
Trace Level   off
Security      ON: Local OS Authentication
SNMP OFF
Listener Parameter File/u01/app/oracle/product/10.1.0/db_1/network/
 admin/listener.ora
Listener Log File/u01/app/oracle/product/10.1.0/db_1/network/
 log/listener.log
Listening Endpoints Summary...

(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=10.10.0.130)(PORT=1521)))
Services Summary...
Service "test" has 1 instance(s).
 Instance "test", status UNKNOWN, has 1 handler(s) for this ser-
vice...
The command completed successfully
```

Don't worry about the status being unknown. That's normal.

TNS Resolution

The listener will sit patiently waiting for connections, but each Oracle client needs a way to identify which database is where. This is done using TNS. TNS configuration is kept in the tnsnames.ora file, which is basically an Oracle equivalent of a host file.

This file can be put in a variety of places. Typically it goes in $ORACLE_HOME/network/admin if you have a full client installed, but each OS has a variety of places it looks when the client starts up. The best way to find out where it's looking is to use `truss` or `strace` while connecting to a dummy database and seeing which files it opens (e.g., `strace sqlplus user/passwsddatabase`).

Here's an example of a tnsnames.ora:

```
testdb, syslogdb =
 (description =
  (address_list =
   (address =
```

```
    (protocol = tcp)
    (host = 10.10.0.130)
    (port = 1521)
   )
  )
  (connect_data =
   (sid = test)
  )
 )
```

What's important here? The first line is the name of the database, followed by all the aliases you want. When you use a client to connect to the database, it'll look for the name of the database you supply in the tnsnames.ora, and when it finds that name it'll use the information associated with it. Because the SID is specified in the description, the name is completely arbitrary.

Inside the description is the important stuff, namely the host, which I recommend being specified as the IP address unless you've got a good reason not to do that. The port will almost always be the default Oracle Listener port 1521. In the connect data section of the description we supply the all-important SID.

You can have as many databases listed in the tnsnames.ora as you like; they all follow the format above. But please note that the file is read completely each time it's accessed, so if you make a syntax error on line 15, line 16 and beyond won't get processed. If something just won't work, check for an error in a database description earlier in the file.

For complete details on syntax, please check out the *Oracle Database Net Services Reference Guide*: http://download-west.oracle.com/docs/cd/B12037_01/network.101/b10776/toc.htm.

Connecting Remotely

Oracle provides the Oracle Instant Client on the Oracle.com download page (see the OCI programming section, below), along with a SQL*Plus client. What's cool is that you can use the small SQL*Plus client instead of installing the full (300+MB) Oracle Client. I've installed both of these packages on my Linux workstation and put the tnsnames.ora file in my home directory as a test (note: this is not the machine the database is on):

```
$ vi ~/.tnsnames.ora
$ sqlplus ben/passwd@testdb
. . .
SQL> select count(*) from sys_log_tbl;
 COUNT(*)
----
903
SQL> quit
```

If you had a full client installed, you would need to make sure your $ORACLE_HOME was set before starting SQL*Plus, but it would work the same way. As was mentioned before, $HOME/.tnsnames.ora is only one of several places you can put it. Be aware that if the client can find a sqlnet.ora, it won't look for the tnsnames.ora file!

For more details on net services, please check out the *Oracle Database Net Services Administrator's Guide*: http://download-west.oracle.com/docs/cd/B12037_01/network.101/b10775/ toc.htm.

Troubleshooting TNS

A nifty tool is *tnsping*, which functions like testing DNS name resolution using ping. When you can't connect to a database via TNS, tnsping comes in handy for figuring out why:

```
bash-2.05$ tnsping testdb
. . .
Used parameter files:
Used TNSNAMES adapter to resolve the alias
Attempting to contact (description = (address_list = (address =
(protocol = tcp) (host = 10.10.0.130) (port = 1521)))
(connect_data = (sid = test)))
OK (10 msec)
```

7. Oracle Programming

The purpose of a database is to store and retrieve data. Oracle provides a wide variety of methods for interacting with the database:

- SQL*Plus for traditional SQL interaction via CLI
- PL/SQL for scripting purposes
- Perl (via the DBI/DBD interfaces)
- a C API called the OCI (Oracle Call Interface), which is used by most Oracle tools
- a C++ interface called OCCI (Oracle C++ Call Interface)
- Pro*C and Pro*C++ precompilers for embedding SQL and PL/SQL in C and C++
- a full complement of Java, .Net, OLE, etc., interfaces

You can find documentation on these interfaces in the Oracle Documentation Library, where you should look at *Application Developer's Guide—Fundamentals* to get started: http://download-west.oracle.com/docs/cd/B14117_01/nav/portal_5.htm.

SQL

SQL can be scripted in the purest sense of the word by simply dumping all your SQL statements into a flat file, naming it [something].sql, and then running it. There is no logic, just SQL as you'd enter it in SQL*Plus. Comments can be added by prefixing them with double dashes.

You can execute your SQL in either of two ways: via the SQL*Plus command line, by preceding the filename with an at sign (@) and leaving off the .sql extension, or via SQL*Plus's non-interactive mode, à la sqlplus user/passwd script, leaving off the .sql just as with the interactive form. Here's an example of the second method:

```
$ cat sample1.sql
— Sample of a simple sql script
CREATE TABLE test (
   i int,
   s char(10)
 );
INSERT INTO test VALUES(1, 'foobar');
INSERT INTO test VALUES(2, 'fooba');
INSERT INTO test VALUES(3, 'foob');
```

```
$ sqlplus ben/passwd
. . .
SQL> @sample1
Table created.
1 row created.
1 row created.
1 row created.
SQL> select * from test where i = 1;
I    S
-----------
1    foobar
SQL> drop table test;
Table dropped.
```

PL/SQL

PL/SQL is a basic scripting language for the Oracle dialect of SQL (based on SQL99). Along with all the typical SQLesque things you can do, PL/SQL contains a helpful dose of basic logic handling and data constructs. It's very similar to ADA in its design.

A PL/SQL script contains as many as three sections: a BEGIN section is required; DECLARE and EXCEPTION sections are optional. Variables use standard SQL data types (a string might be VARCHAR2). In the DECLARE section you declare variables and set up procedures. The BEGIN section is where you put the real meat of your script and where the processing is done. The EXCEPTION section is for handling exceptions (errors). Every script ends with END, signaling the end of execution.

In addition to these sections, you can subclassify a script as a *procedure* or a *function,* able to be reused with packages, etc. Here is an example:

```
-- PL/SQL Example
DECLARE
 acct_balance      NUMBER(11,2);
 acct              CONSTANT NUMBER(4) := 3;
 debit_amt         CONSTANT NUMBER(5,2) := 500.00;
BEGIN
 SELECT bal INTO acct_balance FROM accounts
  WHERE account_id = acct
  FOR UPDATE OF bal;
 IF acct_balance >= debit_amt THEN
  UPDATE accounts SET bal = bal - debit_amt
   WHERE account_id = acct;
 ELSE
  INSERT INTO temp VALUES
   (acct, acct_balance, 'Insufficient funds');
    --insert account, current balance, and message
 END IF;
 COMMIT;
END;
```

As you can see, we initialize three variables in the DECLARE section, and then in BEGIN we process some SQL using PL/SQL-provided logic. PL/SQL code is executed just like SQL statements in a file, as seen in the last section.

PL/SQL itself is too big a subject to discuss in depth here. For more information on PL/SQL, see the *PL/SQL User's Guide and Reference*: http://download-west.oracle.com/docs/cd/B12037_01/appdev.101/b10807/toc.htm.

Pro*C: The Oracle SQL Precompiler

Pro*C is a precompiler for C and C++. You can embed SQL statements in your code, run the precompiler (the proc command, pronounced "pro c") on your source, and out comes a new source file that you can compile as usual. You might think that Pro*C is just for people too lazy to learn the OCI, but it's actually using its own library, SQLLIB, not the OCI. SQLLIB is unsupported (if used without Pro*C) and apparently changes frequently, so it's not advised that you use it directly.

Sysadmins may find the idea of a SQL preprocessor odd, but think about it for a second: All those # lines in your C source are directives for the C preprocessor. When you compile your C source, the preprocessor (cpp) is run on your code before it's fed to the parser and, eventually, the assembler and linker. Pro*C is really just adding a level of preprocessing prior to your calling the compiler. Let's play with a really simple example to get the idea:

```
#include <stdio.h>
#include <sqlca.h> /* SQL Communications Area */
#define UNAME_LEN     20 /* Varchar limits */
#define PWD_LEN 40
VARCHAR username[UNAME_LEN];  /* SQL vars for */
VARCHAR password[PWD_LEN];    /* username and passwd */
int main(){
 strncpy((char *) username.arr, "SCOTT", UNAME_LEN);
 username.len = strlen((char *) username.arr);
 strncpy((char *) password.arr, "TIGER", PWD_LEN);
 password.len = strlen((char *) password.arr);
 EXEC SQL CONNECT :username IDENTIFIED BY :password;
 printf("Connected to the database as user: %s\n", username.arr);
 return(0);
}
```

Points of interest to note here: SQL is executed by using the `EXEC SQL` statement prior to the SQL command. Notice that for the SQL statement, we're using SQL variables using the SQL VARCHAR data type. Probably the most tricky bit to get used to is the mixing of C and SQL variables in the same source. C variables are referred to as "Host Variables" in the documentation. All SQL variables are prepended with a ":" when dereferenced.

It should also be noted that, just as in SQL*Plus, you don't need to capitalize everything, but it's a good way to help you distinguish betwen SQL and "real" C code.

If you're like me, you're probably wondering about "username.len" in the code above. Well, VARCHARs are handled in a weird way; the Pro*C precompiler is actually replacing those statements with structs. Each VARCHAR (and VARCHAR2) struct has two elements, *arr* and *len*. Thus, `VARCHAR username[20];` becomes:

```
struct
{
 unsigned short len;
 unsigned char arr[20];
} username;
```

You'll find several, shall we say, *interesting* little tidbits like this.

To run the precompiler on your code, use the proc binary with at least two arguments, the input source and the output filename:

```
$ proc connect.c connect-pro.c
Pro*C/C++: Release 10.1.0.2.0 - Production on Thu Oct 7 14:59:54
2004
Copyright (c) 1982, 2004, Oracle. All rights reserved.
System default option values taken from:
 /u01/app/oracle/product/10.1.0/db_1/precomp/admin/pcscfg.cfg
$ wc -l connect*.c
 215  connect-pro.c
  24  connect.c
 239  total
```

Once you've successfully precompiled with Pro*C, you can compile your code as usual. Make sure that you link it against the SQLLIB libraries (found in $ORACLE_HOME/precomp/).

Because of all the odd ins and outs of Pro*C, I wouldn't recommend it as a database management interface for common tasks; PL/SQL is much better and easier for that sort of thing. However, if you really want to get down and dirty, Pro*C is probably an easier method than using the OCI directly. Just make sure you expect to spend more time learning and experimenting than coding.

To get the full scoop on Pro*C, read the *Pro*C/C++ Programmer's Guide*: http://download-west.oracle.com/docs/cd/B14117_01/appdev.101/a97269/toc.htm.

The C Interface: OCI

The Oracle Call Interface (OCI) lets you get close and cuddly with Oracle. One of the chief benefits of the OCI is the ability to leverage the OCI Instant Client, a small set of shared libraries that allows your application to run *without* an $ORACLE_HOME defined and thus without a full install of the Oracle client software! This set of libraries (libociei.so, libclnstsh.so.10.1, and libnnz10.so) can be a lifesaver if you're creating small portable applications in places you don't want to install a full client.

However, the OCI isn't for the timid. You can have a look at the headers commonly used: oratypes.h and oci.h.

The typical flow is to initialize the OCI environment (`OCIEnvCreate()`), connect to the database (`OCILogon()`), prepare (`OCIStmtPrepare()`) and then execute (`OCIStmtExecute()`) SQL statements, and finally disconnect (`OCILogoff`). In addition, there is a pile of different OCI handles that you've got to initialize, destroy, and manage, which can become tedious. But because this is a low-level interface, you can go beyond using SQL statements and interact with Oracle directly. Check out a sample SQL select in OCI:

```
text *sqlstmt = (text *)"SELECT * FROM employees WHERE employee_id
= 100";
checkerr(errhp, OCIStmtPrepare(stmthp, errhp, (OraText *)sqlstmt,
   (ub4)strlen((char *)sqlstmt),
   (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, 0, 0,
  (OCISnapshot *) 0, (OCISnapshot *) 0, OCI_DESCRIBE_ONLY));
/* .... */
```

Notice that for safety's sake everything is wrapped in `checkerr()` functions. Here a SQL statement is defined (`text * sqlstmt`) and then prepared with `OCIStmtPrepare()` and executed with `OCIStmtExecute()`. You're seeing a lot of different database handles in there as arguments.

As I said, not for the timid. But if you need absolute power and you've got a lot of time on your hands, OCI is definitely the way to go. Despite the extra work, it fits a variety of common needs and should gradually come to seem easier and easier to work with.

For full details on the OCI, flip through the *Oracle Call Interface Programmer's Guide*: http://download-west.oracle.com/docs/cd/B14117_01/appdev.101/b10779/toc.htm.

Regarding Perl

If you poke into $ORACLE_HOME you'll notice a Perl directory, in which you've got a pretty decent Perl setup at your fingertips. In 10g (10.1.0) you've got Perl 5.6.1, plus a wide variety of applicable modules: mod_perl, URI, Apache, LWP, RPC, and more. Of possibly greatest interest is the inclusion of the OraPERL module and DBI with the DBD for Oracle—proof that Oracle doesn't hate UNIX admins. A quick look at CPAN, however, will tell you that the OraPERL module is intended only for compatibility with Perl 4 apps and that "any new development should use DBI directly."

The included version of the Oracle DBD in Oracle 10.1.0 is 1.12. It's incredibly easy to use, and most sysadmins will probably find the DBD interface far more "homey" than PL/SQL.

To use the Oracle distribution of Perl you'll need to modify two environmental variables. First, add the Oracle library directory to LD_LIBRARY_PATH if you don't have it included in the runtime linkers configuration (*ldconfig* in Linux, *crle* in Solaris).

Second, put the Perl module directories in your PERL5LIB variable so that they are included in INC. If you fail to add these, you're likely to get a slew of errors.

```
$ export LD_LIBRARY_PATH=/u01/app/oracle/product/10.1.0/db_1/\
lib32:$LD_LIBRARY_PATH
$ export
PERL5LIB=/u01/app/oracle/product/10.1.0/db_1/perl/lib/site_\
perl/5.6.1/sun4-
solaris/:/u01/app/oracle/product/10.1.0/db_1/perl/lib/5.6.1
```

If you do forget to set LD_LIBRARY_PATH, you'll notice that the Oracle DBD uses the OCI.

Once you've got these things set up, you can use Perl and the DBI as you'd expect. If you're new to the DBI, I'd strongly suggest picking up the excellent Programming the Perl DBI from O'Reilly. (Tim Bunce, who co-wrote Programming the Perl DBI, is also the author of the Oracle DBI.)

Here's a simple example of using the Perl DBI provided with Oracle10g:

```
#!/u01/app/oracle/product/10.1.0/db_1/perl/bin/perl
# Example Perl DBI/DBD Oracle Example on Oracle 10g
use DBI;
my $dbname = "testdb"; ## DB Name from tnsnames.ora
my $user = "ben";
my $passwd = "passwd";
#### Connect to the database and return a database handle
$dbh = DBI->connect("dbi:Oracle:${dbname}", $user, $passwd);
if($dbh){
 print("Connected as user $user\n");
} else {
 print("Failed to connect!\n");
 exit;
}
#### Prepare and Execute a SQL Statement Handle
my $sth = $dbh->prepare("SELECT owner,table_name,num_rows FROM
all_tables");
$sth->execute();
print("All tables - Got rows:\n");
print("Owner\tTableName\tNumRows\n");
print("---\t-----\t----\n");
while(@row = $sth->fetchrow_array()){
 print("$row[0]\t$row[1]\t$row[2]\n");
}
print("Select Done!...");
#### Disconnect
if($dbh->disconnect){
 print("Disconnected\n");
} else {
 print("Failed to disconnect\n");
}
```

In this script we're grabbing three columns from the data dictionary's ALL_TABLES system table. We connect, grab the rows and output them, and then disconnect from the database when we're done.

If you have trouble connecting to the database, remember that you need to connect through the listener (you can connect locally, but it's pretty fidgety), so double-check that you can properly `tnsping` the database before freaking out about your script.

The output looks like this (several rows removed for clarity):

```
$ ./example.pl
Connected as user ben
All tables - Got rows:
Owner     TableName             NumRows
----------------------------------------
SYS       DUAL                  1
SYS       SYSTEM_PRIVILEGE_MAP  173
SYS       TABLE_PRIVILEGE_MAP   23
. . .
SYS       PLAN_TABLE$
SYS       OLAPTABLEVELS
Select Done!...Disconnected
```

For more information on using the DBI, please refer to CPAN and/or "Programming the Perl DBI": http://search.cpan.org/imb/DBI-1.45/DBI.pm; http://search.cpan.org/imb/DBD-Oracle-1.15/Oracle.pm.

8. SQL*Loader

I discuss the programming interfaces and SQL*Loader for two reasons. First, as a sysadmin, you're the person the database admins are going to bug when they break, and we know more about general programming and parsing techniques than the database admins do (hopefully). Second, as a sysadmin, you need to be really concerned about recovering the database if the system gets dumped/destroyed for some reason. Recovering a table or an index doesn't matter so much; that's stuff the database admins can handle using their normal backups via RMAN. In order to look at recovery systems in the following chapters, we need to put data in the databases, so that we can verify that we did recover the database properly. PL/SQL and, especially, SQL*Loader are good ways to automate some additions to the database (sitting around doing SQL `INSERT`s all day isn't my idea of fun).

SQL*Loader is essentially a fatty parser. It can take delimited input data files and parse and sort them into the correct tables. This is useful for taking output data from some other system (e.g., a firewall log, Apache logs, an off-site tape report) and loading it into Oracle tables.

You can interface with SQL*Loader using the *sqlldr* command line utility or by using PL/SQL. Obviously you need to start by creating a table or tables in which to store the loaded data. Next you'll create a SQL*Loader *control file*. The control file describes the input data, how to load it, and where to put it. Finally, `sqlldr` is executed to connect to the database and load the input data as you described in the control file.

Please note that SQL*Loader control files should not be confused with the database control files we discussed in previous chapters.

Loading Syslog into a Table

As an example, let's load syslog into our database. We'll create a table with three columns: one for the timestamp, one for the hostname, and one for the message.

First, let's create the table, which we'll name "sys_log_tbl":

```
SQL> create table sys_log_tbl (
 2 timestamp date,
 3 hostname varchar2(12),
 4 message varchar2(1024)
 5 );
Table created.
```

Now we'll create the control file for SQL*Loader to use.

```
$ cat syslog.ctl
-- SQL*Loader Control File for Syslog
LOAD DATA
INFILE 'messages.0'
APPEND
INTO TABLE sys_log_tbl
(timestamp     POSITION(01:15) DATE "Mon DD HH24:MI:SS",
 hostname      POSITION(17:21) CHAR,
 message       POSITION(23:1024) CHAR
)
```

The control file contains a number of directives that tell SQL*Loader how to load the data and in which table to load it. The control file we're using is pretty minimalist. You'll notice that the whole thing looks like one long SQL statement, because it is. The statement says: Load data from the input file "messages.0" and append it into the table sys_log_tbl. The statements in parentheses will translate the input file to the column format in the destination table. In this case we're using fixed-field processing using the POSITION keyboard. The first field, "timestamp," is a date occupying characters 1–15, "hostname" is a string occupying characters 17–21 of the input line, and the message is a string that extends from character 23 out to character 1024.

There are a variety of different parsing methods available, but in this case fixed processing works well, since the hostname field is constant. We could tune this even more by including conditionals, but that's beyond the scope of this booklet.

Now we can run SQL*Loader using our control file:

```
$ sqlldr USERID=ben/passwd CONTROL=syslog.ctl LOG=syslog.log
. . .
Commit point reached - logical record count 64
Commit point reached - logical record count 128
Commit point reached - logical record count 192
Commit point reached - logical record count 256
Commit point reached - logical record count 301
```

Now that it's done, let's look at the table to see if everything went where it was supposed to:

```
SQL> select * from sys_log_tbl;
TIMESTAMP          HOSTNAME
-----------------------------
MESSAGE
-----------------------------
05-OCT-04 vixen
genunix: [ID 723222 kern.notice] 00000000fff63cc0 unix:\
sync_handler+12c
(fff57618, 1000000, 1412d55, 0, f0055aa6, f997fe48)
05-OCT-04 vixen
```

```
genunix: [ID 179002 kern.notice] %l0-3: 0000000000000001 \
0000000000000001
000000000001 00000000fff789b8
```

SQL*Loader is generally found to be the fastest method available for loading data into Oracle, followed closely by Data Pump imports.

For more information on SQL*Loader, please refer to Part II of the *Oracle Database Utilities* guide: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10825/toc.htm.

While composing parsing translations in your control files, keep a copy of the *Oracle Database SQL Reference* handy: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10759/toc.htm.

9. Exporting Databases with Data Pump

Oracle, like other RDBMSes, has the ability to import and export databases. In Oracle10g this system was significantly overhauled and dubbed Data Pump. More can be found about this in the *Oracle Database Utilities* manual.

Data Pump provides the ability to export in several modes, including Full Export, Schema, Table, Tablespace, and Transportable Tablespace. Exports are "logical backups," which means the essence of the instance is exported but not the framework.

Exports feel like a real "pack up and go" operation. Think of leaving an office; you don't take the desks or drawers (tablespaces, control files, etc.), you just empty the contents into a big container and leave. The result of an export is a container called a *dump file*. The file is binary and effectively contains a pile of `INSERT` statements which, when imported, update the instance. Because of this you can't import a dump without a database instance ready to accept it, just as you can't unpack your stuff in a new office unless it has a desk and drawers similar to the ones you had before. Therefore an export can't be considered a serious backup method. It's a great way to create a temporary backup or move data from one place to another, but it's not much different from simply writing a Perl or PL/SQL script to output every table into a flat file and then using SQL*Loader to `CREATE` and `INSERT` everything back into tables. "Recovering" a database would involve creating a new instance of the database from scratch (using DBCA, for example) and then importing into that new instance.

Within the limits of Data Pump exports, they do offer some advantages. Because you are importing data into an existing instance, you can easily move tables, tablespaces, schemas, etc., into other databases. Furthermore, because the data is fairly generic, it provides a solid method of migrating from one version of Oracle to another. I've seen several database admins take exports just before applying major patches, as a failsafe.

The Export

The first part of an export is the creation of a directory reference in which to put the dumpfile that will be created by the export utility. This is done with a `CREATE OR REPLACE DIRECTORY` statement. You can name the directory reference anything you like: I call it *exportdir* here. Once it's created, we grant read and write privileges on that directory to the user who will be performing the export, in this case the user *system*:

```
$ echo $ORACLE_SID
test
$ su
Password:
# mkdir /export/oracle_exports
# chown oracle:dba /export/oracle_exports
# exit
$ sqlplus /nolog
. . .
SQL> connect sys/passwd as sysdba;
Connected.
SQL> create or replace directory exportdir as
'/export/oracle_exports';
Directory created.
SQL> grant read, write on directory exportdir to system;
Grant succeeded.
SQL> quit
```

Now that the directory is ready and Oracle can write to it, we can do the export. There are two binaries for exports: the traditional *exp* export utility and the Data Pump version, *expdp*. Both work the same way, but Data Pump offers significantly more features and performance than the traditional tool. (For instance, you can pause import or exports using Data Pump: hit control-c while it's running, and poke around; it won't stop the operation.)

There are more options available than we'll use here, but we'll be performing a FULL export to the specified DUMPFILE and direct logging to the noted logfile. Notice that we write the path in the form *directory:filename.dmp*, where "directory" is the Oracle reference name to the directory we set up in the previous step:

```
$ expdp system/passwd FULL=y \
> DUMPFILE=exportdir:Oracle-Oct11-fullexp.dmp \
> LOGFILE=exportdir:Oracle-Oct11-fullexp.log
Export: Release 10.1.0.2.0 on Monday, 11 October, 2004 17:49
Copyright (c) 2003, Oracle. All rights reserved.
Connected to: Oracle Database 10g Enterprise Edition Release
10.1.0.2.0
With the Partitioning, OLAP and Data Mining options
FLASHBACK automatically enabled to preserve database integrity.
Starting "SYSTEM"."SYS_EXPORT_FULL_01": system/******** FULL=y
 DUMPFILE=exportdir:Oracle-Oct11-fullexp.dmp
 LOGFILE=exportdir:Oracle-Oct11-fullexp.log
Estimate in progress using BLOCKS method...
Processing object type DATABASE_EXPORT/SCHEMA/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 24.68 MB
Processing object type DATABASE_EXPORT/TABLESPACE
Processing object type DATABASE_EXPORT/DE_SYS_USER/USER
Processing object type DATABASE_EXPORT/SCHEMA/USER
. . .
```

```
. . exported "WMSYS"."WM$VERSION_TABLE"                0 KB  0 rows
. . exported "WMSYS"."WM$VT_ERRORS_TABLE"              0 KB  0 rows
. . exported "WMSYS"."WM$WORKSPACE_SAVEPOINTS_TABLE"   0 KB  0 rows
Master table "SYSTEM"."SYS_EXPORT_FULL_01" successfully
loaded/unloaded
******************************************************************
Dump file set for SYSTEM.SYS_EXPORT_FULL_01 is:
/export/oracle_exports/Oracle-Oct11-fullexp.dmp
Job "SYSTEM"."SYS_EXPORT_FULL_01" completed with 2 error(s) at
18:03
```

The export is now complete and ready to be used.

The Import

Once you've got your full export you're good to go. As a test, I've created a clean database instance on my Linux workstation at home and named it *test*, just like the database I exported from at the office.

The first thing we need to do is start up the new clean instance as usual. Once that's done we can copy over the exported dump:

```
$ sqlplus sys/passwd as sysdba;
. . .
Connected to an idle instance.
SQL> startup
ORACLE instance started.
Total System Global Area     159383552 bytes
Fixed Size                   777896 bytes
Variable Size                132915544 bytes
Database Buffers             25165824 bytes
Redo Buffers                 524288 bytes
Database mounted.
Database opened.
SQL> quit;
$ mkdir /u01/app/oracle/DUMPS
$ cd /u01/app/oracle/DUMPS
$ scp hostxyz:/export/.../Oracle-Oct11-fullexp.dmp .
$ sqlplus sys/passwd as sysdba;
. . .
SQL> create or replace directory dump_dir as
'/u01/app/oracle/DUMPS';
Directory created.
SQL> quit
```

Notice that you need to create a directory reference. It doesn't matter what directory or reference name you use, but if you don't create the reference, you'll get errors from Data Pump.

Now let's do the import, using the impdp command:

```
$ impdp system/passwd FULL=y DIRECTORY=dump_dir \
> DUMPFILE=Oracle-Oct11-fullexp.dmp LOGFILE=Oracle-Oct11-import.log
Import: Release 10.1.0.2.0 - Production on Tuesday, 12 October,
2004 1:50
Copyright (c) 2003, Oracle. All rights reserved.
Connected to: Oracle Database 10g Enterprise Edition Release
10.1.0.2.0
With the Partitioning, OLAP and Data Mining options
Master table "SYSTEM"."SYS_IMPORT_FULL_01" successfully
loaded/unloaded
Starting "SYSTEM"."SYS_IMPORT_FULL_01": system/******** FULL=y
DIRECTORY=dump_dir DUMPFILE=Oracle-Oct11-fullexp.
dmp LOGFILE=Oracle-Oct11-import.log
Processing object type DATABASE_EXPORT/TABLESPACE
ORA-31684: Object type TABLESPACE:"UNDOTBS1" already exists
ORA-31684: Object type TABLESPACE:"SYSAUX" already exists
ORA-31684: Object type TABLESPACE:"TEMP" already exists
. . .
Processing object type DATABASE_EXPORT/SCHEMA/DE_POST_SCHEMA_
PROCOBJ...
Processing object type DATABASE_EXPORT/SCHEMA/DE_POST_SCHEMA_
PROCOBJ...
Job "SYSTEM"."SYS_IMPORT_FULL_01" completed with 6877 error(s) at
01:54
$
```

The import completed successfully. Note all the errors (6877 of them!) due to duplication. This isn't a problem; to avoid them, we could have imported with the `TABLE_EXISTS_ACTION=APPEND` parameter.

Finally, let's test the import. Remember, this is a clean instance. All I've done to the database is to create it with DBCA, start it, create a directory reference, and complete the import. I did not create the user *ben* here. If the import worked properly I should be able to log in, since my old user (ben/passwd) can access the "sys_log_tbl" table that we created in the SQL*Loader chapter:

```
$ echo $ORACLE_SID
test
$ sqlplus /nolog
. . .
SQL> connect ben/passwd;
Connected.
SQL> select * from sys_log_tbl;
....
TIMESTAMP      HOSTNAME
-----------------------
MESSAGE
-----------------------
```

```
05-OCT-04 vixen
pseudo: [ID 129642 kern.info] pseudo-device: devinfo0
05-OCT-04 vixen
genunix: [ID 936769 kern.info] devinfo0 is /pseudo/devinfo@0
903 rows selected.
SQL> quit
```

You can find more information on Data Pump in the *Oracle Database Utilities* manual: http://download-west.oracle.com/docs/cd/B12037_01/server.101/b10825/toc.htm.

10. Using RMAN

Why would a sysadmin like RMAN?

- RMAN can perform online (hot) backups.
- RMAN can be used for either partial or complete recovery.
- With RMAN there's no fear of incomplete backups.
- RMAN can perform DBA-initiated backups and recovery without intervention by the sysadmin.
- RMAN integrates with the existing backup infrastructure.

The problem with backing up Oracle using traditional methods is like the problem with backing up file systems. Oracle maintains a huge amount of data in active memory: unless you shut down the database and perform a cold backup you can't be sure that all the transactions and changes have been captured. In order to ensure complete and uncorrupted backups, we need a hot backup method.

Enabling ARCHIVELOG Mode

Most of the high-availability features of Oracle require you to enable ARCHIVELOG mode for your database. When you enable this mode, redo logs will be archived instead of overwritten. The archive logs are stored in a separate location from the tablespaces and can be backed up regularly by your standard filesystem backup (e.g., NetBackup). Archive logs are utilized by RMAN, Data Guard, Flashback, and many others.

If you're going to enable ARCHIVELOG mode on a database that's important to you, I would recommend shutting down the database and doing a cold backup just in case. Keeping a "final noarchivelog mode backup" seems to be a good practice.

Enabling ARCHIVELOG mode is simple: Just connect to your database in mounted but closed mode (startup mount) and alter the database. But if you don't tune a little you'll run into problems down the road: you need to specify `LOG_ARCHIVE_DEST`:

Start by checking the current archive mode:

```
SQL> SELECT LOG_MODE FROM SYS.V$DATABASE;
LOG_MODE
-------------
NOARCHIVELOG
```

We're in `NOARCHIVELOG` mode and we need to change. We can use a database `alter` statement, but that won't be permanent, so let's update the PFILE directly.

The PFILE should be in either $ORACLE_BASE/admin/SID/PFILE or $ORACLE_HOME/admin/SID/PFILE. I'll add the following lines to the end of the file:

```
############################

# Archive Log Destinations -benr(10/15/04)

############################

log_archive_dest_1='location=/u02/oradata/cuddle/archive'

log_archive_start=TRUE
```

Note that you're not required to specify the log destination location, but if you don't, it may end up in a strange place (in my test it went to $ORACLE_HOME/dbs). You can specify as many as 10 different archive log destinations by using the parameters `log_archive_dest_1` through `log_archive_dest_10`. Remember, if you run out of space in your archive log destination the database will *shut down*!

Now we can start up the database in `mount` mode and put it in `ARCHIVELOG` mode:

```
$ sqlplus sys/passwd as sysdba;
. . .
Connected to an idle instance.
SQL> startup mount;
ORACLE instance started.
Total System Global Area     184549376 bytes
Fixed Size                   1300928 bytes
Variable Size                157820480 bytes
Database Buffers             25165824 bytes
Redo Buffers                 262144 bytes
Database mounted.
SQL> alter database archivelog;
Database altered.
SQL> alter database open;
Database altered.
```

You can see here that we put the database in `ARCHIVELOG` mode by using the SQL statement "`alter database archivelog`," but Oracle won't let us do this unless the instance is mounted but not open. To make the change we shut down the instance, then start up the instance again with the `mount` option, which will mount the instance but not open it. Then we can enable `ARCHIVELOG` mode and open the database fully with the `alter database open` statement.

Several system views can provide us with information regarding archives:

V$DATABASE	identifies whether the database is in `ARCHIVELOG` or `NOARCHIVELOG` mode and whether `MANUAL` (archiving mode) has been specified.
V$ARCHIVED_LOG	displays historical archived log information from the control file. If you use a recovery catalog, the

	RC_ARCHIVED_LOG view contains similar information.
V$ARCHIVE_DEST	describes the current instance, all archive destinations, and the current value, mode, and status of these destinations.
V$ARCHIVE_PROCESSES	displays information about the state of the various archive processes for an instance.
V$BACKUP_REDOLOG	contains information about any backups of archived logs. If you use a recovery catalog, the RC_BACKUP_REDOLOG *contains similar information.*
V$LOG	displays all redo log groups for the database and indicates which need to be archived.
V$LOG_HISTORY	contains log history information, such as which logs have been archived and the SCN range for each archived log.

Using these tables we can verify that we are in fact in ARCHIVELOG mode:

```
SQL> select log_mode from v$database;
LOG_MODE
----------
ARCHIVELOG
SQL> select DEST_NAME,STATUS,DESTINATION from V$ARCHIVE_DEST;
```

Learn more about managing archive redo logs in the *Oracle Database Administrator's Guide*: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10739/archredo.htm.

Basic RMAN Backup

Here's a really simple backup that uses RMAN to write its output to a local file instead of the tape subsystem. In this case, we've got our database (SID: cuddle) up and running:

```
$ echo $ORACLE_SID
cuddle
$ rman nocatalog target /
. . .
connected to target database: CUDDLE (DBID=251015092)
using target database controlfile instead of recovery catalog
RMAN> backup database;
Starting backup at 02-NOV-04
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=162 devtype=DISK
channel ORA_DISK_1: starting full datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
input datafile fno=00001 name=/u02/oradata/cuddle/system01.dbf
```

```
input datafile fno=00003 name=/u02/oradata/cuddle/sysaux01.dbf
input datafile fno=00002 name=/u02/oradata/cuddle/undotbs01.dbf
input datafile fno=00004 name=/u02/oradata/cuddle/users01.dbf
channel ORA_DISK_1: starting piece 1 at 02-NOV-04
channel ORA_DISK_1: finished piece 1 at 02-NOV-04
piece handle=/u01/app/oracle/product/10.1.0/db_1/dbs/05g438u6_1_1
comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:01:45
channel ORA_DISK_1: starting full datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
including current controlfile in backupset
including current SPFILE in backupset
channel ORA_DISK_1: starting piece 1 at 02-NOV-04
channel ORA_DISK_1: finished piece 1 at 02-NOV-04
piece handle=/u01/app/oracle/product/10.1.0/db_1/dbs/06g4391f_1_1
comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:03
Finished backup at 02-NOV-04
RMAN> quit
Recovery Manager complete.
```

This is the most basic backup you can do with RMAN. We didn't tell RMAN how or where to back up the database, simply to do it.

The `rman` command is passed two arguments: the first, `nocatalog`, tells RMAN that we aren't using a recovery catalog database, and the second, `target /`, is similar to a SQL*Plus connect statement, with information that RMAN requires to connect to the target database. The target is the database we wish to back up.

Note that RMAN returns some interesting information before giving us a prompt. It confirms that RMAN is connected to the target, and it lists that target. The DBID seen after the target database SID can be very important for later recoveries; you should write it down for future use. RMAN then confirms that because we aren't using a recovery catalog to store backup metadata it will instead store the data in the target database's control files.

The RMAN command `backup database;` sends RMAN on its merry way backing up the database. By default the backup pieces will be placed in the $ORACLE_HOME/dbs directory. This can get very messy, since your system PFILEs are in there too; we recommend that you not use this location for your regular backups.

Two backup pieces were created. The first contains the data files holding the tablespaces, including the undo tablespace. The second backup piece contains the current SPFILE and current control file.

We've opted to use `ARCHIVELOG` mode, which means we can do hot backups. However, we didn't want the hassle and administrative overhead of maintaining a recovery catalog. So here's the rub: Recall that you need a PFILE/SPFILE to start an instance, and you need the control file to point to all the files to be mounted. If the

database were completely destroyed we would need both the SPFILE and the control file to access the backup pieces made by RMAN, but they are *inside* the backup we just made! Nice little loop of confusion, huh? We'll talk about this later, but for now just keep it in mind.

Basic Recovery

To see how RMAN can be useful for recovery, let's take a database and damage it. We'll simulate a tablespace being deleted by a bad script or a stupid DBA and then try to recover the database:

```
$ mv /u02/oradata/cuddle/users01.dbf
/u02/oradata/cuddle/users01.dbf.oops
```

There is our disaster. Let's connect to RMAN and look for suitable backups to recover:

```
$ rman nocatalog target /
. . .
connected to target database: CUDDLE (DBID=251015092)
using target database controlfile instead of recovery catalog
RMAN> list backup;
List of Backup Sets
===================
. . .
BS Key Type   LV Size  Device Type  Elapsed Time   Completion Time
------------------------------------------------------------------
5      Full   528M     DISK         00:01:43       02-NOV-04
BP Key: 5Status: AVAILABLE  Compressed: NO Tag: TAG20041102T134437
Piece Name: /u01/app/oracle/product/10.1.0/db_1/dbs/05g438u6_1_1
List of Data files in backup set 5
File   LV     Type     Ckp         SCN       Ckp Time       Name
-------------------------------------------------------------------
1      Full   1267667  02-NOV-04   /u02/oradata/cuddle/system01.dbf
2      Full   1267667  02-NOV-04   /u02/oradata/cuddle/undotbs01.dbf
3      Full   1267667  02-NOV-04   /u02/oradata/cuddle/sysaux01.dbf
4      Full   1267667  02-NOV-04   /u02/oradata/cuddle/users01.dbf
BS Key Type   LV Size  Device Type  Elapsed Time   Completion Time
------------------------------------------------------------------
6      Full   2M       DISK         00:00:03       02-NOV-04
BP Key: 6 Status: AVAILABLE  Compressed: NO Tag: TAG20041102T134437
Piece Name: /u01/app/oracle/product/10.1.0/db_1/dbs/06g4391f_1_1
Controlfile Included: Ckp   SCN: 1267704   Ckp time: 02-NOV-04
 SPFILE Included: Modification time: 15-OCT-04
```

We can see that we have good, current backups of this database available. Let's now try to recover in the basic way:

```
RMAN> restore datafile '/u02/oradata/cuddle/users01.dbf';
Starting restore at 02-NOV-04
```

```
using channel ORA_DISK_1
channel ORA_DISK_1: starting datafile backupset restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup
set
restoring datafile 00004 to /u02/oradata/cuddle/users01.dbf
channel ORA_DISK_1: restored backup piece 1
piece handle=/u01/app/oracle/product/10.1.0/db_1/dbs/05g438u6_1_1
tag=TAG20041102T134437
channel ORA_DISK_1: restore complete
Finished restore at 02-NOV-04
RMAN> recover datafile '/u02/oradata/cuddle/users01.dbf';
Starting recover at 02-NOV-04
using channel ORA_DISK_1
starting media recovery
media recovery complete
Finished recover at 02-NOV-04
```

Here, because the control files and spfile are intact, we can simply tell RMAN to restore the missing data file, specifying which data file by its fully qualified path (which you can also see in your `list backup;`).

Once the data file is restored, we can *recover* it to confirm that it's consistent. Do not confuse restoration, the act of putting the blocks back on disk, with recovery, the act of bringing the data file into a correct state, similar to fsck'ing a file system:

```
SQL> select TABLESPACE_NAME,STATUS from dba_tablespaces;
TABLESPACE_NAME    STATUS
---------------------------
SYSTEM             ONLINE
UNDOTBS1           ONLINE
SYSAUX             ONLINE
TEMP               ONLINE
USERS              ONLINE
SQL> select FILE#,STATUS,ENABLED,NAME from v$datafile;
FILE# STATUS     ENABLED         NAME
-----------------------------------------------------------------------
1     SYSTEM     READ WRITE      /u02/oradata/cuddle/system01.dbf
2     ONLINE     READ WRITE      /u02/oradata/cuddle/undotbs01.dbf
3     ONLINE     READ WRITE      /u02/oradata/cuddle/sysaux01.dbf
4     OFFLINE    READ WRITE      /u02/oradata/cuddle/users01.dbf
```

Once you're done with the recovery, you'll either want to bring the data file and tablespaces online using `ALTER` statements, or at the very least use SQL*Plus to verify that the tablespaces are online by looking at the Oracle data dictionary. In this case the tablespace is online but we find the data file is offline. Let's fix that by using an `alter` statement:

```
SQL> alter database datafile '/u02/oradata/cuddle/users01.dbf'
online;
Database altered.
```

```
SQL> alter tablespace USERS online;
Tablespace altered.
```

We didn't need to alter the tablespace since it was already online; I did it as a demonstration. Once you've successfully altered the database to bring both the data file and the tablespaces online, you'll want to run the queries above again to double-check.

Listing Backups

Let's spend a minute looking at the history of backups, using the `list` RMAN command. Using the `list backup` RMAN statement, we can list the backups we've made:

```
$ echo $ORACLE_SID
cuddle
$ rman nocatalog
. . .
RMAN> connect target /
connected to target database: CUDDLE (DBID=251015092)
using target database controlfile instead of recovery catalog
RMAN> list backup;
List of Backup Sets
===================
BS Key Type   LV Size  Device Type  Elapsed Time   Completion Time
------------------------------------------------------------------
1      Full   441M     DISK         00:01:23       15-OCT-04
BP Key: 1 Status: AVAILABLE  Compressed: NO Tag: TAG20041015T175207
Piece Name: /export/rman/rman_CUDDLE_01g2k8m8_1_1.bus
List of Data files in backup set 1
File   LV Type         Ckp SCN    Ckp Time  Name
------------------------------------------------------------------
1      Full   396472   15-OCT-04  /u02/oradata/cuddle/system01.dbf
2      Full   396472   15-OCT-04  /u02/oradata/cuddle/undotbs01.dbf
3      Full   396472   15-OCT-04  /u02/oradata/cuddle/sysaux01.dbf
4      Full   396472   15-OCT-04  /u02/oradata/cuddle/users01.dbf
BS Key Type   LV Size  Device Type  Elapsed Time   Completion Time
------------------------------------------------------------------
2      Full   2M       DISK         00:00:02       15-OCT-04
BP Key: 2 Status: AVAILABLE  Compressed: NO Tag: TAG20041015T175207
Piece Name: /export/rman/rman_CUDDLE_02g2k8ou_1_1.bus
Controlfile Included: Ckp SCN: 396502       Ckp time: 15-OCT-04
SPFILE Included:     Modification time: 15-OCT-04
```

Here we can see two backup pieces: the first contains the data files for the "cuddle" database, is 441MB in size, was made to disk, and took 1 minute and 23 seconds to back up. We also see the pieces tag (note the tag is the same for both pieces). Also note that each data file in the backup piece has a Checkpoint System Change Number (Ckp SCN) associated with it (SCNs are covered in Chapter 11, below). The second piece is

2MB in size, took 2 seconds to back up, and was made to disk. Notice that the second piece lists the modification time for the SPFILE and the SCN for the control file.

The `list` command has a number of arguments that allow you to tailor the output. A concise output is seen using `list backup summary`.

See a complete list of options to the RMAN `list` command in the *Oracle Database Recovery Manager Reference* manual: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10770/toc.htm.

Advanced Backup

The method we used for a basic RMAN backup shows how to use RMAN, but it doesn't provide an efficient way to automate the process using traditional tools such as cron. Using a little RMAN scripting, the process can easily be controlled from a script and run from cron or any other scheduling method.

The foundation of this method of using RMAN is built on the *run block*. Within the block is a list of RMAN commands to be run sequentially. When the block is handled by RMAN, it will first verify that each of the input lines in the block is valid and proper and then execute each statement sequentially. This is as good a method as possible for ensuring that the operation is atomic.

Here is a simple RMAN run block (the backup_full.rman we'll use in a minute):

```
run {
 allocate channel d1 type disk;
 backup full database format '/export/rman/rman_%n_%T_%s_%p.bus';
}
```

In this run block we're simply allocating a disk channel and performing a backup to the /export/rman directory using a specific naming convention for the output backup set.

This run block can be run directly from the RMAN prompt by entering it line by line. However, the best way to execute it is from a standard command line, wrapped in a script and/or controlled from cron:

```
$ rman nocatalog target / \
> cmdfile='backup_full.rman' log='/export/rman/rman.log'
RMAN> 2> 3> 4> 5>
```

Here the RMAN executable is called as usual, but we supply the location of an RMAN script to run with the `cmdfile` argument, and we determine the place to output the logging information by the `log` argument.

When RMAN is executing, it will output the RMAN prompts but nothing else. This can be useful for debugging, but should probably be redirected for cleanliness when used from a script or cron.

Comments can be put in RMAN scripts using a hash mark (#).

More details on syntax can be found in the *Oracle Database Recovery Manager Reference*: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10770/.

Advanced Recovery

If you've spent any time with RMAN before reading this booklet, you'll have noticed that it's not really designed for complete disaster recovery. As a sysadmin, I'm concerned about what to do when the entire environment is in ruins and I can't depend on any other system being available. For advanced recovery we're going to look at how you would recover a database if the only parts you have available are the backup pieces. In this case, I'm going to use my two backup pieces from the advanced RMAN backup we just did to recover the database after destroying every trace of it.

Let's review an important point first. RMAN can be utilized using a "recovery catalog," an independent database specifically created to store RMAN's backup information. When used, this database is updated by RMAN with information pertaining to backup pieces and RMAN metadata. One recovery catalog database can be utilized by multiple databases on different systems, for instance, by a "small" installation of Oracle on your backup server.

If we don't use a recovery catalog we're forced to put backup information in the control file. From one perspective, storing backup information in the database control file makes perfect sense, because you store information about all the other resources of your database in there. From another point of view, however, it's a really bad idea, because the control file is one of the files you're backing up! Therefore, if you have to use a control file to store backup information you'll need to keep some things in mind. In particular, if you want to restore the database, you'll need to recover the control file first, either from the RMAN backup pieces (the hard way) or from a filesystem backup of the system. This problem goes away if you have a recovery catalog, because when you start a database restore RMAN can simply ask the recovery catalog for the piece containing the control file and restore it first.

In the following example we will not be using a recovery catalog, and we'll use an SPFILE instead of a regular PFILE, since SPFILEs are the default in 10g.

First we need to check the location of the backup pieces, set the ORACLE_SID (even though there is no database, you must still have a SID) and use RMAN to start the instance for our recovery. Because no PFILE or SPFILE is present it will use the default system parameter file:

```
$ ls -l /export/rman/
total 908326
-rw-r--r-- 1 oracle oinstall 1465      Nov 8 17:31 rman.log
-rw-r----- 1 oracle oinstall 461864960 Nov 8 17:31
rman_TESTINGx_20041108_3_1.bus
-rw-r----- 1 oracle oinstall 2949120   Nov 8 17:31
rman_TESTINGx_20041108_4_1.bus
$ echo $ORACLE_SID
```

```
testing
$ rman nocatalog target /
. . .
connected to target database (not started)
RMAN> startup force nomount;
startup failed: ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file '/u01/app/oracle/prod-
uct/10.1.0/db_1/dbs/inittesting.ora'
trying to start the Oracle instance without parameter files ...
Oracle instance started
Total System Global Area      167772160 bytes
Fixed Size                    1300832 bytes
Variable Size                 115877536 bytes
Database Buffers              50331648 bytes
Redo Buffers                  262144 bytes
RMAN> quit
Recovery Manager complete.
```

The instance is started, so we can now perform the first part of our recovery. As you recall, you need the PFILE or SPFILE in order to properly start an instance, and, since we're not using a recovery catalog, you need the database control file in order to access the RMAN backup information it contains. Since both the SPFILE and control file are *inside* the backup set, we'll need to use the PL/SQL RMAN interface to point RMAN in the right direction.

Here is the PL/SQL you'll need to use:

```
DECLARE
v_dev varchar2(50);   -- device type allocated for restore
v_done boolean;       -- has the controlfile been fully extracted
yet
type t_fileTable is table of varchar2(255)
index by binary_integer;
v_fileTable t_fileTable;     -- Stores the backuppiece names
v_maxPieces number:=1;       -- Number of backuppieces in backupset
BEGIN
-- Initialize the file table & number of backup pieces in the
backup set
-- This section of code MUST be edited to reflect the customer's
available
-- backup set before the procedure is compiled and run. In this
example, the
-- backup set consists of 4 pieces:
v_fileTable(1):='/export/rman/rman_TESTINGx_20041108_4_1.bus';
v_fileTable(2):='/export/rman/rman_TESTINGx_20041108_3_1.bus';
v_maxPieces:=4;
-- Allocate a device. In this example, I have specified 'sbt_tape'
-- as I am reading backup pieces from the media manager. If the
-- backup piece is on disk, specify type=>null
```

```
v_dev:=sys.dbms_backup_restore.deviceAllocate(type=>null,
ident=>'d1');
-- Begin the restore conversation
sys.dbms_backup_restore.restoreSetDatafile;
-- Specify where the control file is to be recreated
 sys.dbms_backup_restore.restoreControlfileTo(cfname=>'/u02/ora-
data/ testing/control01.ctl');

sys.dbms_backup_restore.restorespfileto('/u02/oradata/testing/spfile
');
-- Restore the controlfile
FOR i IN 1..v_maxPieces LOOP
 sys.dbms_backup_restore.restoreBackupPiece(done=>v_done,
handle=>v_fileTable(i), params=>null);
 IF v_done THEN
  GOTO all_done;
 END IF;
END LOOP;
<<all_done>>
-- Deallocate the device
sys.dbms_backup_restore.deviceDeallocate;
END;
/
```

You'll need to edit the array elements of the `v_fileTable` array, including the number of `v_maxPieces` as the number of elements. The `deviceAllocate()` function tells RMAN we're using disk instead of tape. The most important of all are the `restoreControlfileTo()` and `restorespfileto()` functions. The arguments supplied to both indicate where RMAN should put the control file and the SPFILE.

Run this PL/SQL code by putting it in a file named "restore_foundation.sql" and execute it like this (it'll ask for a value—just enter 1; this doesn't affect anything):

```
$ mkdir /u02/oradata/testing
$ vi restore_foundation.sql
```

[Use your favorite editor to make a file of the code.]

```
$ sqlplus / as sysdba @restore_foundation
SQL*Plus: Release 10.1.0.2.0 -- Production on Tue Nov 9 15:19:30
2004
Copyright (c) 1982, 2004, Oracle. All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 -- 64bit
Production
With the Partitioning, OLAP and Data Mining options
Enter value for number: 1
old 10: -- Initialize the file table & number of backup pieces in
the backup set
new 10: -- Initialize the file table 1 of backup pieces in the
backup set
```

```
PL/SQL procedure successfully completed.
SQL> quit
Disconnected from Oracle Database 10g Enterprise Edition Release
10.1.0.2.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options
$ cd /u02/oradata/testing
$ ls -l
total 5618
-rw-r——-       1 oracle      oinstall       2867200 Nov    9
15:19 control01.ctl
-rw-r—r—       1 oracle      oinstall       857     Nov    9
15:19 spfile
```

Now we have the necessary files to start an instance from which to restore the db. `cat` the SPFILE to determine what dump directory paths need to be created for proper startup. Once the directories have been created, you should duplicate the control file so that there are the typical three copies. Finally, you must create a password file for the instance. Moving the SPFILE into $ORACLE_HOME/dbs isn't necessary, but it's a good idea.

```
$ cat SPFILE
 *.background_dump_dest='/u01/app/oracle/product/10.1.0/db_1/admin/
testing/bdump'
*.compatible='10.1.0.2.0'
...
$ cp control01.ctl control02.ctl
$ cp control01.ctl control03.ctl
$ mkdir -p /u01/app/oracle/product/10.1.0/db_1/admin/testing/bdump
$ mkdir -p /u01/app/oracle/product/10.1.0/db_1/admin/testing/cdump
$ mkdir -p /u01/app/oracle/product/10.1.0/db_1/admin/testing/udump
$ orapwd file=/u01/app/oracle/product/10.1.0/db_1/dbs/orapwtesting
\
password=passwd entries=2
$ cp SPFILE
/u01/app/oracle/product/10.1.0/db_1/dbs/spfiletesting.ora
```

Now we've got the meat of our instance ready to be utilized for restoration of the data files. If the instance is currently started, shut it down (`shutdown abort;`) and restart it using the proper SPFILE. The database will be started in `mount` mode, which will start the instance and read the control file(s) but not open the data files:

```
$ rman nocatalog target /
. . .
connected to target database: DUMMY (not mounted)
using target database controlfile instead of recovery catalog
RMAN> startup force mount \
pfile='/u01/app/oracle/product/10.1.0/db_1/dbs/spfiletesting.ora'
Oracle instance started
```

```
database mounted
Total System Global Area     289406976 bytes
Fixed Size                   1301536 bytes
Variable Size                262677472 bytes
Database Buffers             25165824 bytes
Redo Buffers                 262144 bytes
```

If everything has gone well so far, you can now list the backups available and restore the database:

```
RMAN> list backup;
List of Backup Sets
===================
BS Key Type   LV Size  Device Type   Elapsed Time   Completion Time
------------------------------------------------------------------
1      Full   2M       DISK          00:00:01       08-NOV-04
BP Key: 1 Status: AVAILABLE  Compressed: NO Tag: TAG20041108T172608
Piece Name: /u01/app/oracle/product/10.1.0/db_1/dbs/
rman_TESTINGx_20041108_2_1.bus
Controlfile Included: Ckp SCN: 387742       Ckp time: 08-NOV-04
SPFILE Included:      Modification time: 08-NOV-04
BS Key Type   LV Size  Device Type   Elapsed Time   Completion Time
------------------------------------------------------------------
2      Full   440M     DISK          00:01:31       08-NOV-04
BP Key: 2 Status: AVAILABLE  Compressed: NO Tag: TAG20041108T172932
Piece Name: /export/rman/rman_TESTINGx_20041108_3_1.bus
List of Data files in backup set 2
File   LV Type         Ckp SCN    Ckp Time  Name
------------------------------------------------------------------
1      Full   388558   08-NOV-04  /u02/oradata/testing/system01.dbf
2      Full   388558   08-NOV-04  /u02/oradata/testing/undotbs01.dbf
3      Full   388558   08-NOV-04  /u02/oradata/testing/sysaux01.dbf
4      Full   388558   08-NOV-04  /u02/oradata/testing/users01.dbf
RMAN> restore database;
Starting restore at 09-NOV-04
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=160 devtype=DISK
channel ORA_DISK_1: starting datafile backupset restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup
set
restoring datafile 00001 to /u02/oradata/testing/system01.dbf
restoring datafile 00002 to /u02/oradata/testing/undotbs01.dbf
restoring datafile 00003 to /u02/oradata/testing/sysaux01.dbf
restoring datafile 00004 to /u02/oradata/testing/users01.dbf
channel ORA_DISK_1: restored backup piece 1
piece handle=/export/rman/rman_TESTINGx_20041108_3_1.bus
tag=TAG20041108T172932
channel ORA_DISK_1: restore complete
Finished restore at 09-NOV-04
```

At this point you'll need to attempt a recovery of the database. Ordinarily a recovery is performed by applying all the archive logs against the instance; since we don't have any archive logs, however, you'll get an error. But you *must* attempt it; if you do not, you'll be unable to open the database later.

```
RMAN> recover database;
Starting recover at 09-NOV-04
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=160 devtype=DISK
starting media recovery
unable to find archive log
archive log thread=1 sequence=5
RMAN-00571:
===========================================================
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS ============
RMAN-00571: ===========================================
RMAN-03002: failure of recover command at 11/09/2004 15:39:29
RMAN-06054: media recovery requesting unknown log: thread 1 seq 5
lowscn 388558
```

With the restoration and recovery complete, all the data files will be in the proper place. You can now shut down the current instance and start up the database properly. Once the database is mounted you'll need to reset the logs to open the database:

```
RMAN> shutdown immediate;
database dismounted
Oracle instance shut down
RMAN> quit
Recovery Manager complete.
$ echo $ORACLE_SID
testing
$ sqlplus / as sysdba
. . .
Connected to an idle instance.
SQL> startup
pfile=/u01/app/oracle/product/10.1.0/db_1/dbs/spfiletesting.ora
ORACLE instance started.
Total System Global Area       289406976 bytes
Fixed Size                       1301536 bytes
Variable Size                  262677472 bytes
Database Buffers                25165824 bytes
Redo Buffers                      262144 bytes
Database mounted.
ORA-01589: must use RESETLOGS or NORESETLOGS option for database
open
SQL> alter database open resetlogs;
Database altered.
SQL> quit
```

And you're done! You can test your database by querying a couple of tables and connecting as various users. If you don't want to specify the pfile during startup, you can symlink the spfile to $ORACLE_HOME/dbs/init(SID).ora.

After the database is backed up, you'll want to ensure that you either restore from filesystem backups or recreate the listener configuration.

Please note that the PL/SQL interface we used above is *undocumented*: Oracle will not assist you in using it. It also (supposedly) changes between releases. The only documentation that even mentions it is available only if you have a MetaLink account, in DocID 60545.1.

11. Loose Ends

This chapter contains useful information that doesn't fit elsewhere in this booklet. More information on all of these topics can be found at Oracle.com.

Oracle Flashback

Flashback, a feature introduced in Oracle9i and improved (and hyped) in Oracle10g, is similar in concept to the undo feature of your word processor or GIMP/Photoshop. With Flashback you can do something as important as "flashing back" a dropped table or something as minor as undoing your last SQL statement changes to a table. You can even flash back an entire database to an earlier time.

To understand Flashback you need to be clear on two things: the "recycle bin" and Oracle SCNs. A System Change Number, or SCN, is an integer value associated with each change to the database. You might think of revision numbers in a source control system. Each time you perform an action, whether you're adding or removing data, a unique number is associated with the change. Reverting to an earlier state is as easy as telling Flashback which SCN you want to revert to. Obviously, the kink is that if you drop a table the SCN isn't going to help you: therefore Oracle puts dropped objects into a recycle bin rather than blowing them into the nether regions immediately. (This is why dropping an object doesn't reclaim space immediately: to remove objects from the recycle bin, use the `PURGE` SQL statement.)

Before you start playing with Flashback, there is one little catch you need to be aware of: It doesn't work on system tablespaces. This means that if you connect to Oracle as SYS (who uses the system tablespace by default) and create a table, drop it, and then try to flash back to it, the attempt will fail. Flashback works great on a non-system tablespace, but if you blow away a system table you're on your own.

The easiest way to enable Flashback is during database creation with DBCA. As usual, Enterprise Manager makes everything a snap. We'll discuss its setup here in case you want to enable it on existing databases using the SQL*Plus interface.

In order to utilize Flashback you'll need to put your database in `ARCHIVELOG` mode. Then you can set the `DB_FLASHBACK_RETENTION_TARGET` parameter, which defines the duration of retention of flashback logs, and turn Flashback on with an `ALTER DATABASE` statement. Let's look at the setup:

```
SQL> shutdown immediate;
Database closed.
```

```
Database dismounted.
ORACLE instance shut down.
SQL> startup mount;
ORACLE instance started.
Total System Global Area     184549376 bytes
Fixed Size                   1300928 bytes
Variable Size                157820480 bytes
Database Buffers             25165824 bytes
Redo Buffers                 262144 bytes
Database mounted.
SQL> alter database archivelog;
Database altered.
SQL> alter system set DB_FLASHBACK_RETENTION_TARGET=4320;
System altered.
SQL> alter system set DB_RECOVERY_FILE_DEST_SIZE=536870912;
System altered.
SQL> alter system set DB_RECOVERY_FILE_DEST='/u02/fra';
System altered.
SQL> alter database flashback on;
Database altered.
SQL> alter database open;
Database altered.
```

Flashback is now enabled for this database. We've defined a flashback retention of 4320 minutes (72 hours), a recovery file size of 512MB, and a location for the file recovery area (FRA) as /u02/fra.

Let's see Flashback in action now. You can look at the contents of the recycle bin by querying the DBA_RECYCLEBIN table.

```
$ sqlplus ben/passwd
. . .
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Produc-
tion
With the Partitioning, OLAP and Data Mining options
SQL> create table test_table(
 2 id number(2),
 3 name varchar2(30)
 4 );
Table created.
SQL> insert into test_table values (1, 'Ben Rockwood');
1 row created.
SQL> insert into test_table values (2, 'Tamarah Rockwood');
1 row created.
SQL> insert into test_table values (3, 'Nova Rockwood');
1 row created.
SQL> insert into test_table values (4, 'Glenn Rockwood');
1 row created.
SQL> select * from test_table;
```

```
ID       NAME
-------------------------
1        Ben Rockwood
2        Tamarah Rockwood
3        Nova Rockwood
4        Glenn Rockwood
SQL> drop table test_table;
Table dropped.
SQL> select * from test_table;
 select * from test_table
*
ERROR at line 1:
ORA-00942: table or view does not exist
SQL> flashback table "test_table" to before drop;
flashback table "test_table" to before drop
*
ERROR at line 1:
ORA-38305: object not in RECYCLE BIN
SQL> flashback table "TEST_TABLE" to before drop;
Flashback complete.
SQL> select * from test_table;
ID       NAME
-------------------------
1        Ben Rockwood
2        Tamarah Rockwood
. . .
```

In this example I logged in as a user (ben) who by default writes to the users tablespace. Alternately, I could have specified the tablespace explicitly and used the SYS user for testing (i.e., users.table_test instead of table_test). I created a simple table and populated it with some data. Then I dropped the table and verified that it was gone. To restore the table I used the Flashback `to before drop` statement. Another query verified that the table was properly restored.

Note that our first attempt to flash back the table failed. That was because Oracle refers to the table (and thus the recycle bin does too) in all caps and is case-sensitive. The second attempt, giving the table name in all caps, works just fine. If you forget this you'll find yourself repeatedly hitting yourself in the head trying to figure out what went wrong.

For more information about Flashback check out *Oracle Database 10g High Availability with RAC, Flashback and Data Guard* and the *Oracle Database Backup and Recovery Advanced User's Guide* manual: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10734/rcmflash.htm.

Data Guard

Data Guard is Oracle's database replication product, used for creating and maintaining "standby" disaster recovery databases. This is done by leveraging Oracle archived redo logs, which, as you recall, are generated when the database is in archive mode. The

archived redo logs from the primary database are periodically sent to a read-only standby database which applies each archived redo log to itself. Standby databases come in two varieties, physical and logical. A physical standby database is identical block for block to the primary database; hence the standby system design will effectively need to be identical to that of the primary database. A logical standby database can be very different in design from the primary database's as long as it has sufficient resources to contain the primary's data. In the case of a physical standby database, archived redo logs can be applied directly, whereas they need to be ripped apart into SQL statements and applied one by one to a logical standby database. These options allow for a flexible architecture based on your environment's available resources.

The standby database sits idle most of the time in read-only mode, accepting archive redo logs from the primary. This makes the standby system ideal for running reports and intensive queries. If the primary database fails for some reason, the standby can simply be restarted in read-write mode and used until the primary comes back online.

For small organizations where complex architectures for RAC and data warehousing aren't practical, a standby database using Data Guard can fill a variety of needs all at one time.

A brief aside: Some companies use RMAN as a replication scheme by immediately recovering a backup to a standby system and using it for reports and read-only access, just as if they were using Data Guard. RMAN's one advantage over Data Guard for replication is that you are always testing the validity of your backup system. But RMAN is not at all efficient and you typically don't run RMAN backups more than once a day, making it a poor HA solution by itself.

For more information about Flashback check out *Oracle Database 10g High Availability with RAC, Flashback and Data Guard* and the *Oracle Data Guard Concepts and Administration* manual: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10823/toc.htm.

OLTP

Online Transaction Processing (OLTP) databases, as the name implies, handle real-time transactions, which have some special requirements. If you're running a Web store, for instance, you need to ensure that as people order products, updates are made to all affected tables at once. When a product is sold, tables for product inventory, shipping information, account information, and billing must be updated together or the whole transaction could be in jeopardy. Thus OLTP databases must be atomic (an entire transaction either succeeds or fails—there is no middle ground), consistent (each transaction leaves the affected data in a consistent and correct state), isolated (no transaction affects the state of other transactions), and durable (changes resulting from committed transactions are persistent). Achieving all of this can be a fairly tall order but is essential to running a successful OLTP database.

Because OLTP databases tend to be the real frontline warriors as far as databases go, they need to be extremely robust and scalable.

The OLTP feature you tend to hear most often is "row level locking," in which a given record in a table can be locked from updates by any other process until the transaction on that record is complete. This is akin to mutex locks in POSIX threading. In fact OLTP shares a number of the same problems as concurrent programming: When different users or processes could grab for the same thing at the same time, there's the potential to run into problems; for databases, OLTP methodology is the solution.

Several other factors come into play with OLTP databases: the Oracle10g documentation library dedicates a whole section to OLTP. Find more information in the Oracle10g docs: http://www.oracle.com/pls/db10g/portal.portal_demo3?selected=18.

Oracle Options and Extensions

A wide variety of options and extensions for Oracle exist. They include:

RAC Real Application Clusters, formerly Oracle Parallel Server (OPS): Allows for cluster databases

OLAP Online Analytical Processing: "Provides valuable insight into business operations and markets"

Spatial databases: Store geographical information (e.g., GPS coordinates) with records;using Spatial, you can run queries based on geographical parameters, e.g., how many of the given objects in the database are within 50 miles of New York City?

Streams: An information sharing/distribution mechanism for replication and integration (similar to Data Guard)

Ultra Search: A feature of Oracle Collaboration Suite, built on Oracle Database and Oracle Application Server—a Web-based "out-of-the-box" solution that provides searching across multiple repositories

Text: Can perform linguistic analysis on documents, as well as search text using a variety of strategies, including keyword searching, context queries, Boolean operations, pattern matching, mixed thematic queries, and HTML/XML section searching, and can render search results in various formats, including unformatted text, HTML with term highlighting, and original document format;t supports multiple languages and uses advanced relevance-ranking technology to improve search quality

interMedia: Enables efficient management and retrieval of images and audio/video data, including the ability to automate metadata extraction and basic image processing of most popular multimedia formats.

All of these features and more are included in Oracle Enterprise Edition, with the exception of RAC (and RAC extensions), which are Options. RAC is, however, present in the Standard Edition of Oracle. The basic breakdown of Oracle editions is:

Lite Edition: Complete software for building, deploying, and managing mobile database applications

Personal Edition: Full-featured version for individuals, compatible with the entire Oracle Database family; identical to the Enterprise Edition

Standard Edition One: Two-processor version of Standard Edition at an attractive entry-level price

Standard Edition: Four-processor version of Oracle10g, including full clustering support

Enterprise Edition: Industry-leading performance, scalability, and reliability for OLTP, decision support, and content management—the whole enchilada

12. Tools to Lessen the Pain

There are two open source tools that will make Oracle a heck of a lot easier to manage, explore, and interface with: YaSQL and TOra.

YaSQL

YaSQL is Yet Another SQL*Plus replacement. It's a GPL app written in Perl by Jon Nangle and Nathan Shafer.

So what is SQL*Plus missing? History editing is a big one. After typing out a long query and then screwing up a line, you don't want to cut and paste the whole thing back in. PostgreSQL-like listing would be cool, too, to quickly see a list of all tables, indexes, etc. Better output formatting is a must, since SQL*Plus table output is essentially unreadable. And how about bounding! When I look at a big table (or one I think *might* be big), it would be nice to see only the first 10 lines or the last 10 lines, instead of the whole thing, especially when you just want to see a couple of rows to craft a better query. Well, good news: YaSQL does all this and more!

Here's an example that'll sell you instantly:

```
$ export ORACLE_HOME=/u01/app/oracle/product/10.1.0/db_1
$ export ORACLE_SID=cuddle
$ export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH
$ yasql benr/oracle
YASQL version 1.81 Copyright (c) 2000-2001 Ephibian, Inc.
$Id: yasql,v 1.81 2002/03/06 21:55:13 nshafer Exp nshafer $
Please type 'help' for usage instructions
Attempting connection to local database
Connected to: Oracle Database 10g Enterprise Edition Release
10.1.0.2.0 - Prod
auto_commit is OFF, commit_on_exit is ON
benr@cuddle> select COUNT(*) from all_users;
COUNT(*)
--------
24
1 row selected (0.01 seconds)
benr@cuddle> select * from all_users;5
USERNAME  USER_ID       CREATED
--------------------------------
BENR      58            2004-10-14   01:18:18
```

```
SCOTT     57              2004-02-05  13:44:42
MGMT_VIEW 56              2004-02-05  13:37:17
WKPROXY   51              2004-02-05  13:33:46
WKSYS     50              2004-02-05  13:33:46
5 rows selected (0.01 seconds)
benr@cuddle> show all tables;
Table Name             Type     Owner
-----------------------------------
AUDIT_ACTIONS          TABLE    SYS
AW$AWCREATE            TABLE    SYS
...
WM$VERSION_TABLE       TABLE    WMSYS
WM$WORKSPACES_TABLE    TABLE    WMSYS
100 rows selected (0.18 seconds)
benr@cuddle> select * from SYSTEM_PRIVILEGE_MAP; >sys.priv.map.out
173 rows selected (0.10 seconds)
benr@cuddle> !head sys.priv.map.out
PRIVILEGE  NAME                 PROPERTY
----------------------------------------
-3         ALTER SYSTEM         0
-4         AUDIT SYSTEM         0
-5         CREATE SESSION       0
-6         ALTER SESSION        0
-7         RESTRICTED SESSION   0
-10        CREATE TABLESPACE    0
-11        ALTER TABLESPACE     0
```

In this example you can first see a standard SQL statement that counts the rows of the all_users table and finds 24 rows. In the next statement I query for all rows, but append a "5" after the semicolon, telling YaSQL to show me only the first five rows of output. Then I use the `show` YaSQL command to list all the tables available to me (remember my rant about not having this ability, which PostgreSQL provides?). In the last two statements I run a `select` but redirect output to a file and then use `!` to run a system command to read the file from within YaSQL.

That's just a taste of the power of YaSQL. This is a professional-grade interface that is easy to use, flexible, and won't leave you wondering why database admins seem to enjoy needless suffering. It's anything but "Yet Another."

Grab a copy and make sure to mail the authors and tell them how much you value it. There are lots of SQL*Plus replacements, but trust me: look no further.

You can download YaSQL from its SourceForge project page: http://sourceforge.net/projects/yasql/.

TOra

TOra was written by Henrik Johnson and is now owned by Quest Software. It's an open source tool that is available on SourceForge. UNIX versions are available at no

cost, but the Windows version requires a license beyond 30-day use (although this is not enforced). It's written in C++ and based on Qt.

TOra is a graphical interface that allows simple management of your database. The capital TO in TOra stands for "Toolkit for Oracle." Over time, however, it has grown to support PostgreSQL and MySQL as well. It has a SQL editor, table editors, a schema browser, a PL/SQL editor and debugger, a security manager, a server tuning module, and much more.

For the sysadmin, TOra makes the database extremely easy to explore. If you connect to the database as SYSDBA you can scroll up and down the huge list of tables and views and see them in spreadsheet format. This provides an invaluable way to examine the guts of your database without typing out hundreds of SQL queries.

The SQL and PL/SQL editors are pretty sweet, providing syntax highlighting, line stepping, breakpoint debugging, tree parsing, and more. TOra is extremely handy, much better than trial-and-error with VIM and SQL*Plus.

TOra can also manage multiple connections and has a pretty nice login interface, so you can simply click your instance and go. Just make sure when you first try to connect to the database that TNS resolution is working properly (tnsping the instance first) or uncheck "SQL*Net" on the login screen to use a local login.

Binary distributions are available for Win32 and Linux in both RPM and tarball. Some Solaris tarballs have been contributed, and TOra is fully portable for any platform you want it on.

Get more information, look for updates, and check out features and screenshots at the TOra Web site: http://www.globecom.net/tora/.

Appendix: Oracle Processes

Oracle is a big beast, to say the least, composed of a variety of components. Each database is run as an instance. A database server can run multiple instances at a time. Each instance is made up of various components. We can see these as separate processes at the system level:

```
$ ps -ef | grep BEN
  oracle  342  1      0      13:35:59 ?     0:00   ora_pmon_BEN
  oracle  344  1      0      13:35:59 ?     0:00   ora_mman_BEN
  oracle  346  1      0      13:35:59 ?     0:01   ora_dbw0_BEN
  oracle  348  1      0      13:36:00 ?     0:01   ora_lgwr_BEN
  oracle  350  1      0      13:36:00 ?     0:00   ora_ckpt_BEN
  oracle  352  1      0      13:36:00 ?     0:04   ora_smon_BEN
  oracle  354  1      0      13:36:00 ?     0:00   ora_reco_BEN
  oracle  356  1      0      13:36:00 ?     0:02   ora_cjq0_BEN
  oracle  358  1      0      13:36:00 ?     0:00   ora_d000_BEN
  oracle  360  1      0      13:36:01 ?     0:00   ora_s000_BEN
  oracle  366  1      0      13:36:22 ?     0:00   ora_qmnc_BEN
  oracle  368  1      0      13:36:28 ?     0:04   ora_mmon_BEN
  oracle  370  1      0      13:36:28 ?     0:01   ora_mmnl_BEN
  oracle  372  1      1      13:36:31 ?     0:15   ora_j000_BEN
  oracle  512  1      0      15:19:33 ?     0:00   ora_q000_BEN
```

All these processes make up the instance of the database BEN running on this machine. Let's break it down:

pmon: The *process monitor* performs process recovery when a user process fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using. PMON also checks on the dispatcher processes (described below) and server processes and restarts them if they have failed.

mman Used for internal database tasks.

dbw0 The *database writer* writes modified blocks from the database buffer cache to the data files. Oracle Database allows a maximum of 20 database writer processes (DBW0–DBW9 and DBWa–DBWj). The initialization parameter DB_WRITER_PROCESSES specifies the number of DBW processes, n. The database selects an appropriate default setting for this initialization parameter (or may adjust a user-specified setting) based on the number of CPUs and the number of processor groups.

lgwr The *log writer* process writes redo log entries to disk. Redo log entries are generated in the redo log buffer of the system global area (SGA), and LGWR writes the redo log entries sequentially into a redo log file. If the database has a multiplexed redo log, LGWR writes the redo log entries to a group of redo log files.

ckpt At specific times, all modified database buffers in the system global area are written to the data files by DBW*n*. This event is called a *checkpoint*. The checkpoint process is responsible for signaling DBWn at checkpoints and updating all the data files and control files of the database to indicate the most recent checkpoint.

smon The *system monitor* performs recovery when a failed instance starts up again. In a Real Application Clusters database, the SMON process of one instance can perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use and recovers dead transactions skipped during system failure and instance recovery because of file-read or offline errors. These transactions are recovered by SMON when the tablespace or file is brought back online.

reco The *recoverer* process is used to resolve distributed transactions that are pending due to a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions.

cjq0 *Job queue* processes are used for batch processing. The CJQ0 process dynamically spawns job queue slave processes (J000 . . . J999) to run the jobs.

d000 *Dispatchers* are optional background processes, present only when the shared server configuration is used.

s000 Shared server processes.

qmnc A *queue monitor* process monitors the message queues and is used by Oracle Streams Advanced Queuing.

mmon Performs various manageability-related background tasks.

mmnl Performs frequent and lightweight manageability-related tasks, such as session history capture and metrics computation.

j000 A job queue slave (see cjq0).

q000 Oracle Streams AQ or Streams slave process.