The Cuddletech SAs Guide to Oracle

Ben Rockwood

Draft: Feb 10th, 2005

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

Contents

Chapter 1

Introduction

The Oracle Relational Database Management System (RDBMS) has become one of the most powerful and flexable databases avalible. With ever expanding functionality such DataGuard replication and Real Application Clusters (RAC) the divide between database administrators (DBA) and system administrators (SA) is becoming more and more blurred. With the release of Oracle10g this division of responsability has blurred nearly into obscurity. In years past it was interesting to find that the common requirements for sysadmins managing database systems were centered around storage, high avalibility, and tuning to manage the resources needed by the database, and experience with the database itself was meerly a plus. As this wall between DBA and SA collapses its becoming essential that SAs have a solid understanding of Oracle itself in order to effectively interface with DBAs and architects. Currently there are plenty of SAs that can give 6 hour disertations on disk subsystem tuning for databases and tuning for the Oracle SGA but haven't got the foggiest idea of how to extract information for the database itself. This book was written to correct this problem and to provide experienced administrators with an accelerated overview of Oracle itself.

What this book will not cover is database tuning, sizing, or any aspect of system management itself. It is expected that the reader has some ammount of experience with database systems administration, although it is not required. If you know what an SGA is but not how to explore the internals of Oracle, then this book is for you.

The catalyst for this book was rooted in Oracle backups using RMAN. My DBA, as many others do, forbids me from interacting directly with the databse however expected me to provide offsite duplications of the database. This wasn't a problem untill I wanted to test the backups on a sepreate system in order to ensure that proper proceedures were in place should a disaster strike and, as luck would have it, my DBA didn't have time to help me with the RMAN componant. So I set out to understand RMAN, which led me to realize just how little I actually knew the database internals. So, from that simple desire to restore some RMAN offsites spawn this whole book. Even though I "wasted" a

lot of time that could have been better spent on other tasks I've finally gained the added knowladge that makes a world of diffrence when I interface with my DBA.

This book won't teach you everything about Oracle, and its not intended to. There are plenty of "Oracle for Dummies" (or equivelent) books on the market. Instead, your going to get an overview of all the major componants of Oracle, specifically Oracle10g, in a practical start-to-finish linear fashion from chapter to chapter. As a sysadmin I've focused on topics that are of most practical interest to my needs whether it be for practical use or simply to better understand and guide my DBA. Even though you might never export an Oracle database its essential that you understand what it is, so that when your DBA uses them you know what they are doing and how to provide for their requirements. Throughout the book you'll find URL's to Oracle documentation which will provide you with more information on a given topic.

By the end reading this book you'll have the confidence to login to Oracle, configure basic networking, manipulate user access, understand the programming interfaces, utilize major tools for importing and exporting data, interface with the backup system, but most importantly you'll leave with the knowladge required to allow you to learn the database in more detail for yourself. Any mature RDBMS today requires a large amount of knowladge about to simply get you into a position to "poke around" and learn for yourself. Unlike your first UNIX account, its not so easy as just logging in, typing "ls" and learning from there, you need to be able to understand the login proccess, enviromental variables, connection methods, some basic SQL, and possibly some basic network configuration before you can even just explore the internals of the database. The most important thing to take away is the confidence and basic skills required to let you learn for yourself.

1.1 The Relational Model

In this day and age its easy to take the "simplicity" of the modern relational database for granted. When you first learned about relational databases you might have said to yourself "this is just a bunch of spreadsheets!", and more or less you'd be right. However, this wasn't always the case, it was Dr. E.F. Codd, an IBM researcher, that first developed the relational data model in 1970. In 1985, he published a list of 12 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 RDBMS's in general, including Oracle. The following are Codd's rules[1]:

1. Information is represented logically in tables.

2. Data must be logically accessible by table, primary key, and column.

[1] Unable to aquire a copy of Codd's *The Relational Model for Data Base Management: Version 2* this is quoted from O'Reilly's *SQL in a Nutshell* by Kevin & Daniel Kline

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 must be able to define data, views, integrity contraints, authorization, transactions, and data manipulation. [Typically this is SQL.]

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.

Understanding these rules provides you with a fundamental understanding of the relational database and how it is constructed. Of these rules the most difficult one to come to terms with is rule 4, stating that all information describing the database must itself be contained in the database, this is where *system tables* or *data dictionaries* come in and why its so difficult to start exploring the database without a basic understanding of the relation database design rules. Its not terribly diffrent than building a webserver that must be configured using a CGI, while it isn't intiative, it is an admirable design decision.

You by no means need to bother memorizing this list of rules but simply stashing them away in your long term memory will help to clarify some design decisions used by modern relational databases that might otherwise seem odd.

1.2 Play Along at Home

All of the examples and code used throughout this book were done using Oracle 10g Enterprise Edition on Solaris10 (Sun Ultra2 Dual UltraSPARCII workstation, 512M Memory) and Gentoo Linux (AMD AthlonMP Dual 1.2Ghz, 1GB Memory, Kernel 2.6.8.1). The Oracle Database is avalible as a "free" download from Oracle.com, simply agree to the license and download the software. Oracle

provides you with a 30 Day trial period by which to test and explore the product. Plese note that while the software will not stop you from running Oracle longer than that, it is illegal.

Chapter 2

Installation and the OFA

Installation of Oracle is pretty simple. Some parts of it are confused by the intertwining of installation and creation of a database at the same time. In most cases I would personally suggest that you not create a database during installation. Creation is something better left to the Databate Creation Assistant (dbca) after Oracle has already been installed.

Something that can confuse a lot of sysdmins is the layout of files in an Oracle installation. You'll notice that often you'll see installations of Oracle in /u01 and data placed in /u02, /u03, and so on. But if you do a fresh install of Oracle it may not suggest that you install there. So then why do almost all DBAs install Oracle like that?

The answer is the OFA: the Optimal Flexible Architecture (note: optimal, not Oracle). The OFA was created by Cary Milsap in 1991 at the Oracle User's Conference. 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". Other than just creating order from chaos, the OFA ensures that your installation will always be scalable to larger databases, new versions of Oracle, etc.

The installation guidelines, when combining the OFA and the Oracle installation recommendations, are detailed in the following sections.

2.1 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, this group 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 it's primary group must be *oinstall* with *dba* and *oper* as

secondary groups. After Oracle is installed you can change the primary group to *dba*, many people do, but it's not recommended.

2.2 Environmental Vars

Several Oracle environmental variables are utilized by various Oracle tools. Most of these variables should be defined in the databases primary users .profile. The following is an example.

```
umask 022
ORACLE_BASE=/u01/app/oracle
ORACLE_SID=MYSID

ORACLE_HOME=$ORACLE_BASE/product/10.1.0/db_1
ORACLE_PATH=/u01/app/oracle/product/10.1.0/db_1/bin:. #( Oracle version of $PATH)
TNS_ADMIN=$ORACLE_HOME/network/admin  #(Oracle Net Services config files)
```

The most important variables here are ORACLE_SID and ORACLE_HOME. If these aren't defined you'll have problems all over the place, namely when trying to use SQLPlus.

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)

2.3 File Locations

The OFA suggest installing in special top level directories named /u01, /u02, etc. (technically, you can actually name them anything you want, so long as you keep the same idea). Before installing Oracle you should create the directory structure for Oracle and change ownership of the directories to oracle:oinstall. Each top level directory consists of both app and oradata directories (eg: /u01/app, /u02/oradata, ..) where the app directories contain user directories for each database user and the oradata directories contain Oracle datafiles. In this way, your actually installing Oracle into the *oracle* user directory. Therefore, you should set this directory (/u01/app/oracle) as the home directory for the /textitoracle user.

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/ Oracle Software
/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/TAR       Admin Support Logs
```

```
/u01/app/oracle/admin/db_name1/ Admin Subtree for a SID
/u01/app/oracle/doc/            Online Docs
/u01/app/kjf/                   Oracle Home Dir for user kjf
/u01/app/edm/                   Oracle Home Dir for user edm
/u02/oradata/                   Oracle Data
/u02/oradata/db_name1/          Data for SID
```

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

2.4 Memory Parameters

Thanks to the massive memory and IPC requirements of Oracle we need to tune the host OS for Oracle to run. The following are the requirements for Solaris that should be put 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
```

The above values are the Oracle Installer required minimum values. They are, by definition, tunables and therefore can be tweeked upwards for your enviroment but it is not recommended unless you are fully aware of all the possible side effects in doing so.

2.5 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/ × .trc.

2.6 Steps for installation

We can take all this an setup a basic order for installation. They should be:

1. Modify the system tuneables for your platform (/etc/system on Solaris) and reboot to take effect.
2. Create the directory /u01/app/oracle and /u01/oradata
3. Create the groups *dba*, *oinstall*, and *oper*.
4. Create the user *oracle* with the primary group *oinstall* and /u01/app/oracle as the home directory
5. Add the *oracle* user to the *dba* and *oper* groups.
6. Start the Oracle Universal Installer from 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* (The Database Creation Assistant) to create your databases.

When you create your databases with *dbca* continue to use the OFA by installing your datafiles 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 DBAs will request a local mount for /u01 to install oracle, and then two or more /u0x directories with different specs for datafiles. In this way, we can distribute files onto disks with different accesses patterns or tuning specifications.

Chapter 3

Oracle Basics

To get a good understanding of the basics of Oracle, lets set out to do the most basic operations that we would need to really utilize the database. These are:

1. Create the database
2. Connect to and start the database
3. Create a table
4. Add data to a 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

3.1 Create a database

Two methods are available for creating a new database: the *Database Configuration Assistant* (DBCA) GUI application 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 Administrators Guide*.

To use the configuration assistant, export your display (if your not on the console) and run dbca:

```
bash-2.05$ export DISPLAY=10.10.1.100:0
bash-2.05$ pwd
/u01/app/oracle/product/10.1.0/db_1/bin
bash-2.05$ ./dbca
```

We'll then use the following steps in the GUI.

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 *test*
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. I'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", and according to 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 thats 120M for the shared pool, 24M for the buffer cache, 8M for the Large pool, and 24M 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. Finally, 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. Lastly, 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.

```
bash-2.05# cd /u02/oradata/test
bash-2.05# 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 choose "SYSDBA". Once you login you'll get a licensing information screen, agree to the terms to keep playing. After this 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 playing with the Enterprise Manager (EM), go ahead and shutdown the database so we can see how to start it up using SQL*Plus.

3.2 Connect and Startup

Database startup can be done using SQL*Plus or the Enterprise Manager. We'll Use SQL*Plus. There are a variety of ways to login to SQL*Plus, but the one we'll use is the best, specifying the username and password on the command line make it visible using system tools like ps.

```
bash-2.05$ export ORACLE_SID=test
bash-2.05$ sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Oct 5 17:14:46 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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>
SQL> quit
Disconnected from Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - 64bit Producti
With the Partitioning, OLAP and Data Mining options
bash-2.05$ ps -ef | grep -i test
  oracle  1624     1  0 17:16:20 ?        0:00 ora_qmnc_test
  oracle  1612     1  1 17:16:08 ?        0:01 ora_cjq0_test
.......
```

There are a wide range of additional startup arguments that can be made rather than just STARTUP, such as to keep the database from mounting (NOMOUNT) or to force it to start (FORCE).

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

3.3 Create a table

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

```
bash-2.05$ export ORACLE_SID=test
bash-2.05$ sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Oct 5 17:25:27 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect sys/passwd as sysdba
Connected.
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.

SQL>
```

3.4 Add data to a table

Lets 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> insert into address_book
  2  values (3, 'Nova Rockwood', NULL, NULL, 'Menlo Park, CA');

1 row created.

SQL>
```

3.5 Create an index

Index creation works just like you'd expect.

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

Index created.

SQL>
```

3.6 Query the table

Grabbing data from the table is trivial using standard SQL.

```
SQL> select name from address_book;

NAME
------------------------------
```

```
Ben Rockwood
Tamarah Rockwood
Nova Rockwood

SQL>
```

3.7 Creating and Querying a View

Views are simple says to make tables a little more comfortable, especially as they grow large and seemingly out of control.

```
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
Nova Rockwood

SQL>
```

3.8 Destroying Schema Objects

We can easily destroy old or unneeded objects as need be using the DROP statements.

```
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

         3 Nova Rockwood
Menlo Park, CA


SQL> drop table address_book;

Table dropped.

SQL>
```

3.9 Stopping the database

If your not already logged in, do so now and shutdown the database.

```
bash-2.05$ sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Oct 6 11:25:46 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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 different shutdown methods other than immediate. Using shutdown without an argument is rarely used because the database won't shutdown until all connections have been disconnected willingly, during which time it simply won't allow any new connections. The problems is that if a session is hung or a user went to lunch without disconnecting it could take a long long time. Shutdown immediate will safely close out all current connections regardless of whether or not they want to.

3.10 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 using EM (but then most things are). Once your database is in the

proper state, just issuing the SQL "drop test" will destroy the test database we created.

Chapter 4

Poking around inside Oracle

What separates SAs 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 thats a tall order especially without the assistance of the Enterprise Manager or tools like Tora.

I find "smaller" databases like PostGreSQL or 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? \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. Want to see all the systems tables in PostGreSQL? \dS and you see all the systems tables where the configuration info is. When I really started playing with Oracle I was frustrated by the seemingly *blindness* of the whole experience. I could only see the tip of my nose, no further. 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 be able to just browse along them. If you pull up Enterprise Manager and have a look at the list of tables you'll get a total somewhere near (sit down for this): 1468 tables! I think thats worthy of a big "holy $#!". If your thinking "no problem, there are probably a few views that make all the info more usable", you'd be right... except when you list the views you find 3470 of 'em! This immediate answers the "why doesn't SQL*Plus have a \dt feature like PostGreSQL?" question. Thankfully, you can find lists around that have lists of the commonly used tables, such as:

http://www.techonthenet.com/oracle/sys_tables/

Some interesting tables include the ALL_* tables. 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(*)
----------
      1488

SQL>
```

Sadly, when managing Oracle your 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

USERNAME                         USER_ID PASSWORD
------------------------------ ---------- ------------------------------
ACCOUNT_STATUS                   LOCK_DATE EXPIRY_DA
-------------------------------- --------- ---------
DEFAULT_TABLESPACE             TEMPORARY_TABLESPACE           CREATED
------------------------------ ------------------------------ ---------
PROFILE                        INITIAL_RSRC_CONSUMER_GROUP
------------------------------ ------------------------------
EXTERNAL_NAME
--------------------------------------------------------------------------------
DEFAULT                        DEFAULT_CONSUMER_GROUP


SQL>
```

Something you'll immediately find, if you haven't already, is that SQL*Plus table output *sucks*! Look at all that output, it's hard to see whats happening. For this reason, whenever possible limit your queries down to just what you really need. You'll also notice that when you are searching (the where clause)

the search parameter is case sensitive, which can be confusing because so much of SQL is very case insensitive. Most everything is stored in uppercase, so you can either search for 'BENR' or you can use the SQL function *upper()* to convert the casing to uppercase.

Lets try outputting 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

SQL>
```

Must better that time. Whenever possible, just trim down queries as a best practice.

4.1 The Data Dictionary

The Data Directory is what we call the internal tables and views that Oracle stores its own data in. The tables in the data dictionary are broken down into 3 different set 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. The DBA_ views are only intended for admins. And finally, the USER_ views display all the information from the schema of the current user. So the big difference between the USER_ and ALL_ views is that you can only look around in your own schema using USER_ views but the ALL_ views can look at all the schema's the user has permission to. 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

4.2 Essential System Tables

There are a couple of tables that will make your life easier and not make you feel so blind and helpless. This is by no means a complete list, just a couple good ones to get you on your way poking about and learning the lay of the land.

4.3 The Dollar Views

I sometimes hear these mistakenly referred to as the *dollar tables*, but their actually views. These are the system views that are named V$something. Be-

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	Roles 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 grants 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

cause they are in the SYS schema, you'll sometimes see them called properly as SYS.V$something. The views are of interest to SAs because they contain dynamic performance data. When looking at system views you might also see GV$something views, these are for RAC installations were you want to see the Global View.

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

4.4 A word about STATSPACK

We've probably all heard about STATSPACK. The all mystical and power DBA telemetry suite for Oracle. I know that I myself have suggested to DBAs "don't worry, just run STATSPACK" to which I got an icy stare back.

STATSPACK is effectively made obsolete in 10g because so much of the data previously kept by STATSPACK is now kept by the database itself. All the

data is stored in more V$ views and is intended to be viewed via the Enterprise Manager. While graphs aren't as kool as piles of output, I'll take Enterprise Manager in this case.

Chapter 5

Files and Componants

As an SA you need to be able to divide the various files in Oracle into logical groups mentally. Knowing what goes where will help you get a better idea of how you can help maintain Oracle from a systems level. For this discussion, we'll assume that you have installed Oracle according the the OFA discussed earlier.

We can device the installation into 2 main catagories right off the bat: oracle distribution files and instance 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 diffrence between a "database" and an "instance". A "database" is the collection of files that store your data and all the metadata and configuration information to govern how that data is accessed, utilized, etc. An "instance" is a group of proccesses that make your database accessable. So, the datafile "users01.dbf" is a componant of the database and the proccess logwritter proccess "ora_lgwr_SID" is a componant of the instance.

5.1 Database Files

When you create a database you'll get several things.

Control Files Usually 3 named control01.ctl - control03.ctl

Data Files The default *dbca* configuration is to create 5 datafiles: system01.dbf, undotbs01.dbf, sysaux01.dbf, users01.dbf and temp01.dbf.

Redo Logs Typically 3, named redo01.log - redo03.log

Admin Files A series of directories and files usually in $ORACLE_BASE/admin/SID.

A controlfile can be thought of as the Oracle equivelent to a DiskSuite State Database. It contains references and metadata pertaining to all the database files such as datafiles, redo logs, etc. Typically there are 3, or more, copies

of the controlfile, similar to the way SDS uses multiple state databases. This guarrenties that even if 1 control file is lost or corrupt we still have another usable copy. Controlfiles supposedly can be rebuilt if they are lost or destroyed but it isn't easy. If there is a file you want to make sure your backing up, this is it.

The datafiles contain the actual data. These are after refered to as *tablespaces*. Within the tablespace resides all the tables that make up your database. Users use the "users tablespace" which is in the users01.dbf datafile. Temp tables are written to the "temp tablespace" which is in the temp01.dbf datafile. So on and so forth.

Redo logs are the database equivilent to a filesystem journal. When changes are made to the database the changes are written to the redo log. If data were log, we could recover the lost changes by replaying the redo logs just like we'd replay a VxFS journal on an inconsistant filesystem. Redo logging is enabled by default, but can be turned off if you choose by enabling the NOLOGGING parameter. There are typically 3 or more redo logs, which are written to in a round-robin fashion. Each one fills up untill all the logs are full and it begins overwritting the first again.

On a related note that we won't go into depth about, 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 (directory where archived redo logs get stored) becomes full or unwrittable the instance will shutdown! Typically a cron or backup job is written to backup or move the archived redo logs to tape or a permanate storage location on a regular schedual to ensure the archive destination never fills up.

Lets break down the admin files and directories.

bdump Background Dump Directory (Alert Logs and Trace Files)

cdump Core Dump Directory

udump User Dump Directory

create Creation Logs

pfile Paramter File Directory

The *bdump* directory is an important directory because both trace files and alert logs are stored here. If you have a problem with your database the first place you'll want to look is in the alert log.

5.2 Initialization Parameters

Several files contain lists of initialization paramters. The files go by the names *init.ora*, *pfile* (paramater file), or *spfile* (server paramater file). Each database

instance will have a pfile in it's admin directory. An init.ora will exist in the $ORACLE_HOME/dbs/ directory. In the pfile directory of each instances admin directory you 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 (proccesses), size of all the various pools (large_pool_size, etc), and the location of the control files (control_files). The paramter files is read by Oracle when you start an instance, as the name "initialization" implies. Once this file is read and proccessed the control files will point the instance to the rest of the datafiles.

More and more you hear about spfile's. Unlike your init.ora pfile, you can't (well, you shouldn't anyway) hand edit an spfile. Oracle has been making more and more parameters dynamically changable. Apparently in older releases of Oracle changing parameters would involve shutting down the database, modifying the paramter in the pfile and then restarting. Now many of these paramters can be change with an ALTER statement and take effect immediately without a downtime. The problem was that if you changed these dynamic paramters the changes wouldn't be persistant across database restarts, so the DBA would have be careful to always update the pfile after making the change. Obviously this became a problem quickly as careless DBAs would simply forget. So to combat the problem the spfile was made avalible where dynamicly changed parameters would be recorded in the "binary" spfile and thus make tunable paramters persistant. To use an spfile you actually need to explicitely create one by converting your existing pfile into a pfile (eg: "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 like an spfile is created on your behalf, whereas in previous releases it had to be manual created as noted above.

```
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.

5.3 Oracle Startup

The best way to understand how all the diffrent files work together is to examine the Oracle startup proccess. Even though it might seem like a strange place in

this book to cover the topic, it makes the most sense from a sysadmin point of view, so lets dive in.

When Oracle starts an instance it reads the spfile or pfile to determine the initialization paramters. It uses these paramters to allocate the SGA and create background proccesses. All this is done without associating a database to the instance! At this point the instance is started but not mounted, or as some say "Started in no mount mode". They say that 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 datafiles are actually associated is the instance. It's somewhat akin to loading the bullets into a gun; the gun is mearly a vehicle to utilize bullets and without them it's utterly useless. As a fun test, you can rename the datafile directory (where the control and datafiles are) and startup the instance without mounting. You won't get an error, you won't get a complaint... 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 revolation is extremely important. Why? You'll notice that alot of the documenation, particularly reguarding 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 thats been destroyed. Well, now you know.

Moving along to the second phase of database startup, the database is mounted. In this step we associate the control, data, redo, and other database

related files to 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.

Lets quickly look at what happens when you mount your database with the instance already running but with the all the data and control files missing (renamed 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 controlfile and not the datafiles. Thats because the parameter file has a listing of all the controlfiles and the controlfile is responsable for storing information about all the other datafiles and resources used by the database. If the controlfile can't be read the database doesn't know what else exists! This is why you should be careful to keep good backups of your controlfiles using plain ol' system backups. This is also why Oracle maintains multiple copies (typically 3) of the control file for safety.

Lets put all the datafiles back and try mounting the database again.

```
# mv test.HOLD/ test
# sqlplus sys/passwd as sysdba
SQL*Plus: Release 10.1.0.2.0 - Production on Wed Oct 13 14:36:09 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

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

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

Once an instance is started using a pfile and the mount proccess has used the controlfile(s) to associate the datafiles with the instance we need to *open* the database. Untill a database is opened it is not accessable. It's equivilent to starting a system in single-user mode. Some ammount of interaction with the databse is avalible 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).

But here's the confusing part, the normal data dictionary tables (ALL_USERS, for example) will give you an error, but most of the V$ tables, which are supposed to be the dynamic tables, work fine! What exactly "fixed" is supposed to mean in the case I dunno.

```
SQL> select * from all_users;
select * from all_users
              *
ERROR at line 1:
ORA-01219: database not open: queries allowed on fixed
 tables/views only
SQL> select status from v$instance;

STATUS
------------
MOUNTED
SQL>
```

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

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

The shutdown proccess is the simply opposite of the startup.

```
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
```

5.4 Oracle Distribution Files

The Oracle distribution all the stuff written to disk during the install proccess (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, create databases, back them up, manage them, etc.

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 the were installed, and how. The Oracle Inventory doesn't impact databases in any way and only impacts the installer. You can see a list of all the products and options you have installed in the *oraInventory/Components* directory.

Chapter 6

Users and Permissions

Security and Permissions are, I think, probly the most confusing topics pertaining to Oracle. The line between the UNIX user (oracle) and the Oracle users (sys, etc) gets really blurred. Just getting logged into SQL*Plus can be a major task because you aren't sure whether it wants UNIX or Oracle user names and passwords. I'll try and demystify this a bit.

6.1 Users and Passwords

A big part of the confusion over authentication is due to the fact that there are *two* diffrent forms of authentication! 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 into the database without authenticating to Oracle based on your UNIX UID. 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 step 10 by selecting "All Initialization Parameters" and then selecting "Show Advanced Parameters"! If you scroll down the long list you'll see "os_roles" is set as false, which you can then enable if you choose. All this tells you one thing: Oracle isn't keen on you using it. Therefore... don't.

When you create a database with *dbca*, in step 5, it will ask you to assign passwords to the default Oracle accounts. The default accounts are: sys, system, dbsnmp, and sysman. Lets break down the default users. (Descriptions taken directly from the DBCA help.)

SYS The SYS user owns all base tables and user-accessable view of the data dictionary (Oracle configuration information). No Oracle user should ever alter (update, delete, or insert) any rows or schema objects conatained in the SYS schema, because such activity can compromise data integrety. The security administrator must keep strict control of this central account.

SYSTEM The SYSTEM user is used to create additional tables and views that display administrative information, and internal tables and views used by various Oracle options and tools.

SYSMAN The SYSMAN user represents the Enterprise Manager super admin account. This EM admin can create and modify other EM admin accounts as well as admin the database instance itself.

DBSNMP The DBSNMP user is used by EM to monitor the database. EM uses this account to access performance stats about the database. The DBSNMP credentials sometimes referred to as the monitoring credentials.

In addition to these users, a user can connect with diffrent levels of privileges, namely SYSDBA and SYSOPER. When you connect using "connect sys/passwd as sysdba" your connecting as the SYS user and requesting SYSDBA privs. Because the SYS user is the Oracle equivilent to 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 privs:

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

The big diffrence between SYSDBA and SYSOPER privs is that SYSDBA can do anything (just like root). The SYSOPER privs allow you just about the same amount of control but won't allow you to look at user data. Both privs allow you to ALTER DATABASE, CREATE SPFILE, STARTUP or SHUTDOWN, ALTER DATABASE ARCHIVELOG, and includes RESTRICTED SESSION privs. However, only SYSDBA can CREATE or DROP DATABASE, and the ALTER DATABASE RECOVER options for SYSOPER are limited to complete recovery only.

Naturally these are all administrative accounts and therefore shouldn't be used for general database usage. (We used *sys* in our basics chapter because its already there and I wanted to focus on the basics without jumping straight into authentication).

6.2 Adding New Users

If you really poke around with Oracle you'll find that the *sys* account is blocked from a variety of tasks therefore we need to create some users. The easiest way to add users is with the Enterprise Manager, if you can use it.

The following is an example of adding a use using SQL:

```
bash-2.05$ sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Fri Oct 8 11:19:42 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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.

SQL>
```

So here, as SYSDBA we've created the user "ben" with the default profile identified by the password "passwd". The users default tablespace is users and temp 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 the quota on the users tablespace by user ben.

Now lets login with the user...

```
bash-2.05$ sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Fri Oct 8 11:24:11 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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'
SQL>
```

Wait? Even though the user is added the user has no privs? This is indeed lame. The user wasn't defined with any *roles*. Each user must have one or more security role assigned to them. You'll notice that even though in the creation of the user our second SQL statement was to grant the "CONNECT" role to the user, but that wasn't enough. By default in 10g a new user has no privs at all, so we need to just keep piling on the roles. Lets spend a minute sorting out the diffrence between privileges, roles, and profiles.

6.3 Privileges, Roles and Profiles

At first the three can be confusing to differentiate. Lets 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 ammount of control over the user here, by bundling privileges into roles and 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, so lets look at roles in more depth.

Lets list some of the predefined roles that Oracle makes avalible 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. Includes: 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. Also the following roles: 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 privileges) and the following roles: 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 avalible and DBA_ROLE_PRIVS contains the user-to-role mappings. If your 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
....
```

Like we used when creating the user, we can use the GRANT SQL 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

SQL>
```

6.4 Other Permissioning Problems

You can sometimes run into wierd problems with users ability to modify the database. Here's an example.

```
SQL> insert into test_tbl
  2  values (1, 'Some data');

1 row created.

SQL> select * from test_tbl;

     FIRST SECOND
---------- ------------------------------
         1 Some data

SQL> create table test_tbl_2 (
  2  first number(2),
  3  second varchar2(30)
  4  );
create table test_tbl_2 (
*
ERROR at line 1:
ORA-01536: space quota exceeded for tablespace 'USERS'

SQL>
```

WTF? I can update a table, and I have the CONNECT role granted to my user but I can't write due to a quota? I thought that using the "default" profile would keep this sorta thing from happening!

I removed the user from the RESOURCE role, and as soon as I did I lost the ability to create tables, even though the CONNECT role explicitely allows CREATE TABLE... and why a quota error and not a permissions error?

....

6.5 Changing Passwords

Passwords can be changed by altering the user. Mind you, SYSDBA needs to do it.

```
SQL> ALTER USER "BENR" IDENTIFIED BY "passwd";

User altered.
```

Chapter 7

The Listener

The listener is the Oracle componant that (duh) listens for connections. It allows remote connection to the database, and naturually a database that can only be used locally isn't very interesting. Its part of the larger *Oracle Net Services* framework.

Just an FYI: In the past Oracle also used the marketing terms "Net8" and "SQL*Net" too. It's the same basic thing. Some diffrences exist but very few. See the diffrences here: *http://www.orafaq.com/faqnet.htm*

There are 3 diffrent interfaces to 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 you can do it without these tools.

7.1 Configuring the Listener

Configuring the listener is easy enough. The listener are 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 defines our essential parameters, namely the SID_NAME (same as $ORACLE_SID) and ORACLE_HOME (same as $ORACLE_HOME). The name "SID_LIST_LISTENER" actually is derived from the LISTENER name, so if you did actually change the listeners 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 diffrent 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

7.2 Starting the Listener

The listener can be started before or after the database instance is started, it doesn't matter. You can start the listener with a simple *lsnrctl start*:

```
bash-2.05$ 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 service...
The command completed successfully
bash-2.05$
```

Don't worry about the status being unknown. Thats normal. (Not 100% sure why, but it is.)

7.3 TNS Resolution

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

This file can be put in a variety of places. Typically it should go in *$ORACLE_HOME/network/admin* if you have a full client installed, but each OS has a variety of diffrent 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 (ie: "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)
              )
          )
```

Ok, it looks alittle strange, I admit. It's just a lot of nested statements, and actually reminds me alot of Solaris Zones configuration.

Whats important here? First line is the name of the database, followed by all the aliases you want. It's important to note that you can use any name 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. Name the database "mysqlsbetter" if you want.

Inside the description is the important stuff, namely the "host" which I recommend as being the IP address unless you've got a good reason not too. 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 above format. 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 proccessed. If something just won't work, maybe it's 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

7.4 Connecting remotely

Something really kool that Oracle provides is the *Oracle Instant Client* on the Oracle.com download page (we mention this later in the OCI programming section). Optionally you can also download and install a SQL*Plus client to go with it. Whats so kool is that you can use SQL*Plus on a client without installing the full Oracle Client (300MB+ install). I've installed both of these packages on my Linux workstation and put the *tnsnames.ora* file listed in the last section in my home directory as a test (note, this is not the same machine the database is on):

```
[benr@nexus benr]$ vi .tnsnames.ora
[benr@nexus benr]$ sqlplus ben/passwd@testdb
SQL*Plus: Release 10.1.0.2.0 - Production on Mon Oct 11 14:37:46 2004
Copyright (c) 1982, 2004, 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

SQL> select count(*) from sys_log_tbl;

  COUNT(*)
----------
       903

SQL> quit
[benr@nexus benr]$
```

If you had a full client installed you would need to make sure your *$ORACLE_HOME* was set before starting SQL*Plus but it'd work the same way. As mentioned before, *$HOME*/.tnsnames.ora is just one of several places you can put it. Something to be aware of is 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

7.5 Troubleshooting TNS

There is a nifty tool that you'll want to know about: *tnsping*. It does what you'd expect, and is basically like testing DNS name resolution using ping instead of

a more appropriate tool. When you can't find a database via TNS it's pretty handy for figuring out why.

```
bash-2.05$ tnsping testdb
TNS Ping Utility for Solaris: Version 10.1.0.2.0 -
 Production on 11-OCT-2004 16:09:53
Copyright (c) 1997, 2003, Oracle.  All rights reserved.
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)
```

Chapter 8

Oracle Programming

The point of a database is to store and retrieve data. These basic utilizations of a database are essentially application specific. Your not generally gonna interact with your database by firing up SQL*Plus every time you need to get a phone number from your address book.

Oracle provides a wide variety of diffrent methods of interacting with the database. There is obviously SQL*Plus for traditional SQL interaction via CLI. We can also use PL/SQL for scripting purposes. PERL can be used via the DBI/DBD interfaces. There is a C API called the OCI (Oracle Call Interface) which is used by most Oracle tools. There is a C++ interface called OCCI (Oracle C++ Call Interface). Plus there is a full compliment of Java, .Net, OLE, blah blah blah interfaces if your into that sorta thing. Your C and C++ apps can also utilize Pro*C and Pro*C++ precompilers for embedding SQL and PL/SQL in your code.

Lets take a look at some of the interfaces. We've already used SQL*Plus in the last chapter so we'll skip that one.

You can find documentation on these interfaces in the Oracle Documentation Library, where you should particularly have a look at *Application Developer's Guide - Fundamentals* to get started:

http://download-west.oracle.com/docs/cd/B14117_01/nav/portal_5.htm

8.1 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 "–".

You can execute your SQL in one of two ways, via the SQL*Plus command line, by proceeding the filename with an and leaving off the implied *.sql* extension, or via SQL*Plus's non-interactive mode, ala: *sqlplus user/passwd script*, leaving off the *.sql* just as with the interactive form.

Here's an example:

```
oracle@nexus6 PLSQL$ cat sample1.sql
-- Sample of a simple sql script
-- Non-PL/SQL
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');
INSERT INTO test VALUES(4, 'foo');
INSERT INTO test VALUES(5, 'foobar');
INSERT INTO test VALUES(6, 'fobar');
INSERT INTO test VALUES(7, 'fbar');
INSERT INTO test VALUES(8, 'bar');
oracle@nexus6 PLSQL$ sqlplus ben/passwd
 (....)
SQL> @sample1
Table created.
1 row created.
1 row created.
1 row created.
1 row created.
1 row 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.
```

8.2 PL/SQL

PL/SQL is a basic scripting language for Oracle dialect of SQL (based around SQL99). Along with all the typical SQL-eque things you can do there is added a helpful dose of basic logic handling and data constructs to let you do all the normal things you'd expect. It's very similar to ADA in its design.

A PL/SQL script contains 1 to 3 sections. At the least you need a BEGIN section, and optionally DECLARE and EXCEPTION sections. Variables use

standard SQL datatypes (a string might be VARCHAR2). In the DECLARE section you can declare variable or setup procedures. In the BEGIN section you you put the real meat of your script, this is where the proccessing is done. The EXCEPTION section is for handling exceptions (errors). Every script ends with "END" signalling the end of execution.

In addition to these sections, you can subclassify scripts as a *procedure* or a *functions*. This allows for re-usablity, used in associate 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;
```

You can see that in the DECLARE section we initialize 3 variables, 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.

You'll notice that variables are assigned values with ":=". In fact, if you haven't looked at any ADA lately, I'll toss in an ADA example, look a little familiar?

```
-- ADA95 Example (Not Oracle)
procedure OneInt is
   Index : INTEGER; -- A simple Integer type
begin
   Index := 23;
   Put("The value of Index is");
   Put(Index);   -- The default field width is 11 columns
   New_Line;
   Index := Index + 12;
   Put("The value of Index is");
```

```
   Put(Index, 8);
   New_Line;
end OneInt;
-- Result of execution
-- The value of Index is         23
-- The value of Index is      35
```

PL/SQL itself is a big subject, and we won't bother to touch it here. Damn near everything an SA could want to do can be done in a plain ol' SQL file. Any SA would likely rather use PERL than PL/SQL, but there are piles and piles of books devoted to PL/SQL out there.

Find more PL/SQL information in the *PL/SQL User's Guide and Reference*: *http://download-west.oracle.com/docs/cd/B12037_01/appdev.101/b10807/toc.htm*

8.3 Pro*C: The Oracle SQL Precompiler

Pro*C, with it's kool name and swanky asterisk, is a precompiler for C and C++. You can embed SQL statements in your code and then run the precompiler (the command is *proc*) on your source and out comes a new source file that you can compile as usual. The interesting thing is that while at first you might think that Pro*C is just for people to lazy to learn the OCI that it's actually using its own library SQLLIB, not the OCI. SQLLIB is unsupported (if used without Pro*C) and apparently changes frequently so they don't advise that you use it directly.

At first glance most sysadmins and developers might be confused by the idea of a SQL preproccessor... but think about it for a second: all those # lines in your C source are directives for the C Preprocessor. If you recall, when you compile your C source one of the first things that happens is the preproccessor (cpp) is run on your code before it's feed to the parser and eventually the assembler and linker. So Pro*C is really just adding another level of preproccessing prior to your calling the compiler.

Lets just 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);
}
```

Things 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 datatype. Probly the most tricky thing to get used to is the mixing of C and SQL variables in the same source. C variables are refered to as "Host Variables" in the documentation. All SQL variables are prepended with a ":" when deferenced.

It should be noted that you don't need to capitolize everything, just like in SQL*Plus, but it's a good idea to help you keep SQL vs "real" C code seperated in your mind.

If your like me, your probly looking at that example code above and wondering whats up with "username.len". Well, VARCHARs are handled in a wierd 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 precomipiler on your code, use the *proc* binary with at least 2 arguments, the input source and the output filename.

```
bash-2.05$ 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
```

```
bash-2.05$ wc -l connect*.c
     215 connect-pro.c
      24 connect.c
     239 total
bash-2.05$
```

Once you've successfuly 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 tasts, 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 probly an easier method than using the OCI directly... just make sure you expect to spend more time learning and experiementing 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*

8.4 The C Interface: OCI

The Oracle Call Interface (OCI) lets you get close and cuddly with Oracle. One of the chief benifits of the OCI is the ability to leverage the *OCI Instant Client*, a small set of shared libaries that allow your application to run *without* an $ORACLE_HOME defined... and thus without a full install of the Oracle client software! These libraries (libociei.so, libclnstsh.so.10.1, and libnnz10.so) can really be a life saver if your 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 enviroment (*OCIEnvCreate()*), connect to the database (*OCILogon()*), prepare (*OCIStmtPrepare()*) and then execute (*OCIStmtExecute()*) SQL statments, and finally disconnect (*OCILogoff*). Along with that, you've got a pile of diffrent OCI handles that you've got to initialize, destroy, and manage which can become tedious. However, because this is a low level interface you can work outside of just 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 sake everything is wrapped in checkerr() functions. Here a SQL statement is defined (text * sqlstmt) then it is prepared with OCIStmtPrepare(), and executed with OCIStmtExecute(). Your seeing alot of diffrent database handles in there as arguments.

Like I said, not for the timid. But if you need absolute power and you've got way too much time on your hands, OCI is definately the way to go. If fits a variety of common needs despite the extra work and after spending some time with the API should get easier and easier to work with.

For the full detail 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

8.5 Regarding PERL

If you poke your head into $ORACLE_HOME you'll notice a PERL directory. If you dig around you'll find that 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 including: mod_perl, URI, Apache, LWP, RPC, and more. Of possibly more interest are 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 compatability with PERL 4 apps and *"any new development should use DBI directly."*

The included version of the Oracle DBD is 1.12, in Oracle 10.1.0. Its increadably easy to use and most SAs will probly find the DBD interface far more "homey" than PL/SQL.

To use the Oracle distribution of PERL you'll need to modify 2 enviromental variables. Firstly you'll need to add the Oracle library directory to LD_LIBRARY_PATH if you don't have it included in the run time linkers configuration (*ldconfig* in Linux, *crle* in Solaris). Secondly you'll need to put the PERL module directories in your PERL5LIB variable so that they are included in INC. If you fail to add these your likely to get a slew of errors.

```
$ export LD_LIBRARY_PATH=/u01/app/oracle/product/10.1.0/db_1/lib32
$ 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, however, forget to set LD_LIBRARY_PATH you'll notice interestingly that the Oracle DBD uses the OCI.

Once you've got things setup, you can use PERL and the DBI like you'd expect. If your new to the DBI I'd strongly suggest picking up the excellent book *Programming the PERL DBI* from O'Reilly. (Insidently, 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 the above script we're grabbing 3 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 figity) and ensure that you can properly tnsping the database before freaking out about your script.

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

```
bash-2.05$ ./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
bash-2.05$
```

For more information on usign the DBI please refer to CPAN and/or *Programming the PERL DBI.*

http://search.cpan.org/~timb/DBI-1.45/DBI.pm

http://search.cpan.org/~timb/DBD-Oracle-1.15/Oracle.pm

Chapter 9

SQL*Loader

You might wonder, why in this guide do I want to talk about the SQL*Loader... I mean, isn't this an SA centric paper? Why care about DBA utilities. Well, I discuss the programming interfaces and SQL*Loader for two reasons. First, as an SA the DBAs are gonna bug *you* when they break, and we know more about general programming and parsing techniques than the DBAs do (hopefully). Secondly, as an SA I'm really concerned about recovering the database if the system gets dumped/destroyed for some reason. I don't really care about recovering a table or an index, thats stuff the DBAs can handle using their normal backups via RMAN hooked into our awsome NetBackup server. In order to look at recovery systems in the following chapters we need to actually put data in the databases so that we can verify we actually did recover the database properly. Thus, PL/SQL and SQL*Loader in particular are good ways to automate some additions to the database so we can later recover databases with some feeling that we really got it all back. Sitting around doing SQL INSERTs all day ain'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 tables we want them in. This is useful for taking output data from some other system (a firewall log? apache logs? offsite tape report?) and loading it into Oracle tables. So, basically, it is what it sounds like.

You can interface with SQL*Loader using the *sqlldr* command line utility or using PL/SQL. Obviously you first need to start by seting up a table for the data to go. Next you'll create a SQL*Loader control file. The control file describes the data, how to load it and to where to put it. Then we can actually go and run the loader on our data and put it in the database.

9.1 Loading SYSLOG into a table

As an example, lets load something suffiently SA-ish into our database... syslog. We'll create a table with 3 columns: one for the timestamp, one for the system

name, and one for the message.

First, lets create the table. We'll use a real user this time now that we know how to create users.

```
bash-2.05$ sqlplus ben/passwd
SQL*Plus: Release 10.1.0.2.0 - Production on Fri Oct 8 13:04:21 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
SQL> create table sys_log_tbl (
  2  timestamp date,
  3  hostname varchar2(12),
  4  message varchar2(1024)
  5  );

Table created.
SQL>
```

Ok, tables ready. Now the control file for SQL*Loader to use.

```
bash-2.05$ cat syslog.ctl
-- SQL*Loader Control File for Syslog
-- Oct  5 11:28:33 vixen su: [ID 810491 auth.crit]....
-- Oct  5 11:29:06 vixen last message repeated 4 times
-- 1-----  -0----| |--0| |--------------->
-- 1----------->15 17-21 23--->1024
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
)
bash-2.05$
```

The control file contains a number of directives that direct SQL*Loader to do the right thing. The one we're using is pretty minimalist. You'll notice that the whole thing looks like one long SQL statement, because it is. The statement, if we read it outload say: Load data from the input file 'messages.0' and append it into the table sys_log_tbl. The statements in paranthasis following read like a table column discription. These statements will translate the intput file to the column format in the destination table. In this case we're using fixed field proccessing using the POSITION keyboard. If we read this we get: the first field "timestamp" is a date occupying charrectors 1-15, hostname is string of

charrectors occupying charrenctors 17-21 of the input line, and finally message is a string of charrectors that extends from the 23rd charrector out to 1024 charrectors.

There are a variety of diffrent parsing methods avalible other than using fixed proccessing, but in this case it works well. We could tune this even more including the use of conditionals to better parse the input but thats beyond the scope of this book.

And now we can actually run SQL*Loader using our config file.

```
bash-2.05$ sqlldr USERID=ben/passwd CONTROL=syslog.ctl LOG=syslog.log

SQL*Loader: Release 10.1.0.2.0 - Production on Fri Oct 8 17:06:28 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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
bash-2.05$
```

Now that it's done, lets look at the table to see if it all ready went where it was supposed to.

```
bash-2.05$ sqlplus ben/passwd

SQL*Plus: Release 10.1.0.2.0 - Production on Fri Oct 8 17:09:23 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

SQL> select * from sys_log_tbl;
......
TIMESTAMP HOSTNAME
--------- ------------
MESSAGE
--------------------------------------------------------------------------------
unix: [ID 100000 kern.notice]

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
.......
```

This should give you some good ideas about how to quickly load data into your database. SQL*Loader is generally found to be the fastest method avalible 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

For reference 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

Chapter 10

Exporting databases with Data Pump

Oracle, like other RDBMS', 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 frame work.

As an SA, exports feel like a real "pack your shit 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 (dump file) and leave. The result of an export is a *dump file*. The file is binary and effectivly just contains a pile of INSERT statements that when imported updates 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 into a new office unless it has a desk and drawers similar to the one you just left. Therefore you can't really look at an export as a serious backup method. It's a great way to cover your ass or to move data from one place to another, but it's not much diffrent than just writting 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 invole creating a new instance of the database from scratch (using *dbca* for instance) and then importing into that new instance.

With the limits of Data Pump exports understood, there are some advantages to exports. 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 or Oracle to another (if needbe). I've seen several DBAs take exports just before applying major patches, just as a failsafe.

10.1 The Export

The first part of an export is the creation of a directory reference to put the dumpfile the 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, here I call it "exportdir". Once its created, we grant read and write privs on that directory to the user that will be performing the export, in this case the user *system*.

```
bash-2.05$ echo $ORACLE_SID
test
bash-2.05$ su
Password:
# mkdir /export/oracle_exports
# chown oracle:dba /export/oracle_exports
# exit
bash-2.05$ sqlplus /nolog
SQL*Plus: Release 10.1.0.2.0 - Production on Fri Oct 8 11:35:26 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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 actually 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 avalible 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 setup in the previous step.

```
bash-2.05$ 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

bash-2.05$
```

The export is now complete and ready to be used.

10.2 The Import

Once you've got your full export your 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 thats done we can copy over the exported dump.

```
oracle@nexus6 ~$ sqlplus sys/passwd as sysdba;
SQL*Plus: Release 10.1.0.2.0 - Production on Tue Oct 12 01:31:02 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
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;
oracle@nexus6 ~$ mkdir /u01/app/oracle/DUMPS
oracle@nexus6 ~$ cd /u01/app/oracle/DUMPS
oracle@nexus6 DUMPS$ scp hostxyz:/export/.../Oracle-Oct11-fullexp.dmp .
oracle@nexus6 DUMPS$ sqlplus sys/passwd as sysdba;
SQL*Plus: Release 10.1.0.2.0 - Production on Tue Oct 12 01:48:48 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> create or replace directory dump_dir as '/u01/app/oracle/DUMPS';
Directory created.

SQL> quit
```

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

Moving onwards, lets do the actual import using the *impdp* command.

```
oracle@nexus6 DUMPS$ 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=Or
acle-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
```

```
oracle@nexus6 DUMPS$
```

Ok, the import completed sucessfully. You'll notice all the errors (6877 of them!) due to duplication. This isn't a problem, we could have imported with "TABLE_EXISTS_ACTION=APPEND" parameter to avoid these.

Finally, lets 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 done the import. I never created the user "ben". So, if the import worked properly I should be able to log in as my old user (ben/passwd) can access the "sys_log_tbl" table that we created in the SQL*Loader chapter.

```
oracle@nexus6 DUMPS$ echo $ORACLE_SID
test
oracle@nexus6 DUMPS$ sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Oct 12 01:55:59 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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;
Disconnected from Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options
oracle@nexus6 DUMPS$
```

And it works!

10.3 Digging Deeper

I hope you can see that Data Pump provides some very useful functionality but it's not a quick and dirty backup method that an SA could use without the interaction of a DBA.

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

Chapter 11

Using RMAN

SysAdmin's have a definate love/hate relationship with RMAN. Many of us not so jokingly refer to it as the "Recovery Mangler". You love it because its fairly simple and is the only sure-fire way to perform online backups. You hate it because it's often cryptic, and either works or doesn't. Because RMAN couples closely with a backup infastructure there is alot of finger pointing when problems arise between "It's an RMAN issue!" and "No, it's a NetBackup issue!". RMAN causes DBAs and UNIX admins to get too close for comfort and niether generally understands the wholistic view of the interaction properly.

So, why would a sysadmin like RMAN?

- RMAN can perform online (hot) backups
- RMAN can allow for partial or complete recovery
- No fear of incomplete backups
- DBA initiated backups and recovery without the interaction of the SA
- Intigration with existing backup infistructure

The problem with backing up Oracle using traditional methods is similar to the problems with backing up filesystems, unless you shutdown the database and perform a cold backup there is no way to know that all the transactions and changes have been written to datafiles. The SGA maintans a huge amount of data in active memory which can cause a problem. It's alittle like editing a configuration file on the system and then halting the system and wondering where your changes went. In order to ensure consistance of the database we need a hot backup method. If we restore a filesystem backup of the database that was taken while Oracle was running we run the risk of lossing database changes at best or having a corrupt database at worst.

11.1 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 archivelogs are stored in a seperate place usually can backed up regularly by your standard filesystem backup system (NetBackup or whatever). Archive logs are utilized by RMAN, Data Guard, Flashback and many others.

If your going to enable archivelog mode on a *real* database thats 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 and excepted practice.

Enabling archive 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 alittle you'll run into problems down the road, so lets specify some parameters too. Namely, consider LOG_ARCHIVE_DEST.

Lets start by checking the current archive mode.

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

LOG_MODE
------------
NOARCHIVELOG
```

So we're in NOARCHIVELOG mode and we need to change. We can use a database alter statement, but that won't be perminant, so lets just 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 we're not actually required to specify the location of the log destination, but if you don't it'll end up in strange places (in my test it went to $ORACLE_HOME/dbs making a mess). You can specify as many as 10 diffrent archive log destinations by using the paramters 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 startup the database in mount mode and put it in archivelog mode.

```
[oracle@vixen pfile]$sqlplus sys/passwd as sysdba;
SQL*Plus: Release 10.1.0.2.0 - Production on Fri Oct 15 16:00:58 2004
```

```
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
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 shutdown the instance, and then startup the instance again but this time 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.

There are several system views that can provide us with information reguarding archives, such as:

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 infact 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

11.2 Basic RMAN Backup

Lets do a real simple backup using RMAN that writes it's output to a local file instead of the tape subsystem just to see how it works. In this case, we've got our database (SID: cuddle) up and running.

```
[oracle@vixen oracle]$ echo $ORACLE_SID
cuddle
[oracle@vixen oracle]$ rman nocatalog target /
Recovery Manager: Release 10.1.0.2.0 - 64bit Production
Copyright (c) 1995, 2004, Oracle.  All rights reserved.

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.
[oracle@vixen oracle]$
```

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

The *rman* command is passed 2 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 backup.

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

The RMAN command *backup database;* sends RMAN on its merry way backing up the database. Notice that we didn't tell it where or how to backup the data. By default the *backup peices* will be placed in the $ORACLE_HOME/dbs directory. This can get very messy since your system PFILES are in there too, and therefore we recommend that you don't use this location for your normal backups.

Two backup peices were created. The first contains the datafiles holding the tablespaces including the undo tablespace. The second backup peice contains the current SPFILE and curent controlfile.

Lets stop and think carefully for just a moment. Now, 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 is the rub: recall that you need a PFILE/SPFILE to start an instance and you need the controlfile to point to all the files to be mounted. If the database were completely destroyed we would certainly need both the SPFILE and the Controlfile to access the backup peices 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.

11.3 Basic Recovery

To see how RMAN can be useful for recovery, lets take a database and damage it. Lets simulate a tablespace being deleted because of a bad script or stupid DBA and then try to recover the database.

```
[oracle@vixen oracle]$ mv /u02/oradata/cuddle/users01.dbf /u02/oradata/cuddle/users01.dbf.oc
```

Okey, there is our disaster. Lets connect to RMAN and look for suitable backups to recover.

```
[oracle@vixen oracle]$ rman nocatalog target /
Recovery Manager: Release 10.1.0.2.0 - 64bit Production
Copyright (c) 1995, 2004, Oracle.  All rights reserved.

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: 5   Status: AVAILABLE  Compressed: NO  Tag: TAG20041102T134437
        Piece Name: /u01/app/oracle/product/10.1.0/db_1/dbs/05g438u6_1_1
  List of Datafiles 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

RMAN>
```

We can see that we have good and current backups of this database avalible. Lets now try to recover in the basic way.

```
MAN> 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

RMAN>
```

Here, because the controlfiles and spfile are in tact, we can simply tell RMAN to restore the missing datafile, specifying which datafile by it's fully qualified path (which you can also see in your "list backup;").

Once the datafile is restored, we can recover it to ensure it's consistant.

Once your done, you'll either want to bring the datafile 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.

```
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
```

In this above case the tablespace is online but we find the datafile is offline. Lets just fix that up 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 because it was already online, but I did it any way to demonstrate. Once you've successfully altered the database to bring both the datafile and the tablespaces online you'll want to run the queries above again to double check.

11.4 Listing Backups

Lets spend just a minute looking alittle more at the history of backups using the "list" RMAN command. Using the "list backup" RMAN statement we can list the backups we've made.

```
[oracle@vixen oracle]$echo $ORACLE_SID
cuddle
[oracle@vixen oracle]$rman nocatalog
Recovery Manager: Release 10.1.0.2.0 - 64bit Production
Copyright (c) 1995, 2004, Oracle.  All rights reserved.

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 Datafiles 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

RMAN>
```

Here we can see 2 backup pieces, the first contains the datafiles for the "cuddle" database, is 441MB in side, was made to disk and took 1 minute and 23 seconds to make. We aslo see the peices tag (notice the tag is the same for both peices). Notice also that each datafile in the backup peice has a Checkpoint System Change Number (Ckp SCN) associated with it (SCNs were covered earlier). The second peice is 2MB in size, took 2 seconds to backup and was done to disk. Notice that the second peice lists the modification time for the SPFILE and the SCN for the Controlfile.

The *list* command has a number of argument that can allow you to tailor the output to just about any way you want to see it. A nice and consice 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

11.5 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 proccess using traditional tools like the cron. Using alittle RMAN scripting the proccess can easily be controlled from a script and run from cron or any other schedualling method you'd like.

The foundation of this method of using RMAN is built on the *run block*. Within the block are a list of RMAN commands to be sequentually run. When the block is handled by RMAN it will first verify that each of the input lines in the block are valid and proper, it will then execute each statement line by line sequentually. This is as close a method as possible to ensure 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 preforming a backup to the */export/rman* directory using a specific naming convension for the output backup set.

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

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

Here the *rman* executable is called as it would normally be, but we supply the location of an RMAN script to run with the *cmdfile* argument and a place to output the logging information with the *log* argument.

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

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

More details on command line options can synatex for run blocks can be found in the Oracle Database Recovery Manager Reference:

http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10770/

11.6 Advanced Recovery

If you've spent any time with RMAN previous to reading this book you'll have noticed that it's not really built for complete disaster recovery. As a sysadmin, I'm concerned about what I do when the entire enviroment is in ruin and I can't depend on any other system being avalible. So, for the advanced recovery we're going to examine how you would recover a database is the only thing you have avalible is the backup pieces. In this case, I'm going to use my 2 backup peices from the advanced RMAN backup we just did to recover the database after destroying every trace of the database.

Lets review an important point first. RMAN can be utilized using a recovery catalog. When used, this database is updated by RMAN with information pertaining to backup peices and RMAN metadata. One recovery catalog database can be utilized by multiple databases on diffrent systems, possibly 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 some other place... the controlfiles. Storing backup information inside the database controlfile is a real touchy subject. On one hand it makes perfect sense because your store information about all the other resources of your database in there anyway. On the other hand, its an insanely stupid idea because the controlfile is one of the files your backing up! Therefore, if you have to use a controlfile to store backup information you'll need to keep some things in mind. Particularlly, if you want to restore the database you'll need to recover the controlfiles first either from the RMAN backup peices (the hard way) or even possibly recover it from a file system backup of the system before recovering the backup peices made by Oracle. All this goes away if you have a recovery catalog because when you start a database restore RMAN can simply ask the recovery catalog for the peice containing the control file and restore it first.

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

So, to start off we need to double 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:

```
[oracle@vixen oracle]$ 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
[oracle@vixen oracle]$ echo $ORACLE_SID
testing
[oracle@vixen oracle]$ rman nocatalog target /
Recovery Manager: Release 10.1.0.2.0 - 64bit Production
Copyright (c) 1995, 2004, Oracle.  All rights reserved.
```

```
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/product/10.1.0/db_1/dbs/inittestir
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.
[oracle@vixen oracle]$
```

Ok, the instance is started and we can now preform the first part of our recovery. If you recall from earlier discussions you need the PFILE or SPFILE in order to properly start and instance, and we need the database controlfile in order to access the RMAN backup information it contains because we're not using a recovery catalog. Since both the SPFILE and controlfile are *inside* the backup set we'll need to use the PL/SQL RMAN interface to specifically point RMAN in the right direction.

Here is the PL/SQL you'll need to use (*restore_foundation.sql*):

```
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
-- Initialise the filetable & number of backup pieces in the backupset
-- This section of code MUST be edited to reflect the customer's available
-- backupset before the procedure is compiled and run. In this example, the
-- backupset 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:=2;

-- Allocate a device. In this example, I have specified 'sbt_tape' as I am
```

```
-- reading backuppieces from the media manager. If the backuppiece 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 controlfile is to be recreated
sys.dbms_backup_restore.restoreControlfileTo(cfname=>'/u02/oradata/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), par
        IF v_done THEN
                GOTO all_done;
        END IF;
END LOOP;

<<all_done>>
-- Deallocate the device
sys.dbms_backup_restore.deviceDeallocate;

END;
/
```

The parts of this code you'll need to edit are the array elements of the *v_fileTable* array including the number of *v_maxPieces* as the number of elements. Then the deviceAllocate() function tells RMAN we're using disk instead of tape. But the most important lines are the *restoreControlfileTo()* and *restorespfileto()* functions. The arguments supplied to both will indicate where RMAN should put the controlfile and 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 mean anything):

```
[oracle@vixen RMAN]$ mkdir /u02/oradata/testing
[oracle@vixen RMAN]$ vi restore_foundation.sql
[oracle@vixen RMAN]$ 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: -- Initialise the filetable & number of backup pieces in the backupset
new  10: -- Initialise the filetable 1 of backup pieces in the backupset

PL/SQL procedure successfully completed.

SQL> quit
Disconnected from Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - 64bit Producti
With the Partitioning, OLAP and Data Mining options
[oracle@vixen RMAN]$ cd /u02/oradata/testing
[oracle@vixen 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
[oracle@vixen testing]$
```

Now we have the neccisary files to start an instance to restore from properly. *cat* the spfile to deterime what paths need to be created for proper startup, namely the dump directories. Once the directories are created you should duplicate the controlfile so that there are the typical 3 copies. And finally, you must create a password file for the instance. Moving the spfile into *$ORACLE_HOME/dbs* isn't neccisary, but a good idea.

```
[oracle@vixen testing]$ cat spfile
*.background_dump_dest='/u01/app/oracle/product/10.1.0/db_1/admin/testing/bdump'
*.compatible='10.1.0.2.0'
...
[oracle@vixen testing]$ cp control01.ctl control02.ctl
[oracle@vixen testing]$ cp control01.ctl control03.ctl
[oracle@vixen testing]$ mkdir -p /u01/app/oracle/product/10.1.0/db_1/admin/testing/bdump
[oracle@vixen testing]$ mkdir -p /u01/app/oracle/product/10.1.0/db_1/admin/testing/cdump
[oracle@vixen testing]$ mkdir -p /u01/app/oracle/product/10.1.0/db_1/admin/testing/udump
[oracle@vixen testing]$ orapwd file=/u01/app/oracle/product/10.1.0/db_1/dbs/orapwtesting pas
[oracle@vixen testing]$ 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 a real restoration of the datafiles. If the instance is currently started, shut it down (*shutdown abort;*) and restart the instance using the proper SPFILE. The database will be started in *mount mode* which will start the instance and read the controlfile(s) but not actually open the datafiles.

```
[oracle@vixen testing]$ rman nocatalog target /
Recovery Manager: Release 10.1.0.2.0 - 64bit Production
Copyright (c) 1995, 2004, Oracle.  All rights reserved.

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

RMAN>
```

If everything has gone well so far, you can now list the backups avalible 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 Datafiles 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>
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

RMAN>
```

At this point you'll need to attempt a *recovery* of the database. Normally a recovery is preformed by applying all the archivelogs against the instance, but since we don't have any archivelogs we'll get an error instead. Even though you'll get an error you *must* attempt it anyway, 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

RMAN>
```

With the restoration and recovery complete all the datafiles will be in the proper place. You can now shutdown the current instance and startup 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.
[oracle@vixen testing]$ echo $ORACLE_SID
testing
[oracle@vixen testing]$ sqlplus / as sysdba
SQL*Plus: Release 10.1.0.2.0 - Production on Tue Nov 9 15:31:45 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.

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 your done! You can test our your database by querring a couple tables and connecting as various diffrent users. If you don't want to be bugged with specifying the pfile during startup you'll just need to symlink the spfile to *$ORACLE_HOME/dbs/init(SID).ora*.

After the database is back 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* and Oracle will not assist you in using it. It also (supposedly) changes between releases. Unfortunetly, this is the only way. The only documentation that even mentions it is avalible only if you have a MetaLink account, in DocID 60545.1.

Chapter 12

Loose Ends

This chapter contains information that I don't think fits anywhere else in this book but that I none-the-less would like to cover. More information on all of these topics can be found at Oracle.com.

12.1 Oracle Flashback

FlashBack is a feature introduced in Oracle9i and improved (and hyped) in Oracle10g. Effectively, it's a "oh shit!" proctection mechanism for DBAs. FlashBack leverages Oracle undo segments (remember the undo tablespace undo01.dbf?). Undo segments used to be called *rollback* segments, and you might have heard that muttered before.

To be more plain, Flashback is similar in idea to the Undo feature of your word proccessor or GIMP/Photoshop. You work along happily and then suddenly realize you really don't like where things are going, so rather than having to fix it we can just Undo. Applications in the last several years actually allow you use an Undo history to undo things you did several changes ago. Flashback is the same concept but for the database. Once Flashback is enabled and you have been granted permission to use it you could do something as important as "flashbacking" a dropped table, or something as minor as undoing your last SQL statements chnages to a table. You can even flash back an entire database to get back to an earlier point in time across the board!

To understand Flashback you need to be clarified on two things: The "recycle bin" and Oracle SCNs. An *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 do something, whether your 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. Because of this you won't

reclaim space immediately when you drop an object, however you can forcably *purge* objects from the recyle bin using the "PURGE" SQL statement.

The componants needed for Flashback have actually been in database for awhile to facilitate OLTP. All OLTP changes need to be atomic (discussed later) so when a transaction is modifying the database and for some reason fails (or in dba speak "throws an exception") the transactions that were uncommited are *rolled back*. *Rollback Segments*, now called undo segments, provided the neccisary historical information to allow for this. All this is leveraged, repackaged and dubbed "Flashback".

Before you get started playing with flashback, there is one little catch you need to be aware of: it doesn't work on the 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 flashback it, it will fail. Flashback works great on the non-system tablespace, but if you blow away a system table your going to take more extreme measures, not just flashback restore it.

The easiest way to enable flashback is to enable it during database creation with *dbca*. And, as usual, Enterprise Manager makes everything a snap. We'll discuss it's 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 that defines the period of time which we want to retain flashback logs, and finally turn Flashback on with an ALTER DATABASE statement. Lets 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.
```

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

Lets see Flashback in action now. You can look at the contents of the recyle bin by querying select DBA_RECYCLEBIN the table.

```
oracle@nexus6 ~$ sqlplus ben/passwd
SQL*Plus: Release 10.1.0.2.0 - Production on Thu Nov 4 00:41:36 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
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, 'Hunter Rockwood');
1 row created.

SQL> select * from test_table;
        ID NAME
---------- ------------------------------
```

```
         1 Ben Rockwood
         2 Tamarah Rockwood
         3 Nova Rockwood
         4 Hunter 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
         3 Nova Rockwood
         4 Hunter Rockwood

SQL>
```

In this example we logged in as a user (ben) that by default writes to the users tablespace. Alternately, we could have specified the tablespace explicitly and used the sys user for testing (ie: users.table_test instead of table_test). I've created a simple table and populated it with some data. Then we drop the table and verify that it's really gone. To restore the table we simply use the flashback "*to before drop*" statement. Another query verifies that is was properly restored.

I want you to specically notice that our first attempt to flashback the table failed! This is because Oracle refers to the table (and thus the recycle bin does too) in all caps and is case sensative. The second attempt using the table name in all caps works just fine. If you forget this you'll find yourself repeatedly hitting yourself in the head to figure out what went wrong.

For more information about Flashback check out the book *Oracle Database 10g High Availability with RAC, Flashback and Dataguard* 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

12.2 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 *arhived redo logs* 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 that applies each archived redo log to itself each time ones comes over. Standby databases come in 2 varieties: Physical and Logical. A physical standby database is identical block for block to the primary database, hense the standby system will need to be designed effectively identical to the primary database. A logical standby database can be very diffrent in design to the primary database so long as it has suffient resources to contain the primaries data. In the case of a physical standby archived redo logs can be directly applied to the standby database whereas the archived redo logs when applied to a logical standby need to be ripped apart into SQL statements and applied one by one. The options allow for a flexable architecture based on your enviroments avalible 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 idea 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 untill 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, I'll note that some companies actually 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. The advantage of RMAN over Data Guard for replication is that you always are testing the validity of your backup system. The downside to using RMAN is that it's not at all effectient and you typically don't run RMAN backups more than once a day making it a poor HA solution by itself. There are a variety of opinions on this practice but just know that it isn't unheard of.

For more information about Flashback check out the book *Oracle Database 10g High Availability with RAC, Flashback and Dataguard* and the *Oracle Data Guard Concepts and Administration* manual:

http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10823/toc.htm

12.3 OLTP

Databases tend to get split up into a variety of diffrent catagoies based on their application and requirements. All of these diffrent catagories naturally get nifty buzz words to help classify them and make distinctions in features more apparent. The most popular buzz work (well, acronymn anyway) is *OLTP* or *Online Transaction Proccessing*. Other classifications include *Descision Support Systems* (DSS), *Data Warehouses*, *Data Marts*, etc.

OLTP databases, as the name implies, handle real time transactions which inherently have some special requirements. If your running a store, for instance, you need to ensure that as people order products they are properly and effiently updating the inventory tables while they are updating the purchases tables, while their updating the customer tables, so on and so forth. OLTP databases must be atomic in nature (an entire transaction either succeeds or fails, there is no middle ground), be consistant (each transaction leaves the affected data in a consistant and correct state), be isolated (no transaction affects the states of other transactions), and be durable (changes resulting from commited transactions are persistant). 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 front line warriors, as far as databases go, they need to be extremely robust and scalable to meet needs as they grow. Whereas an undersized DSS database might force you to go to lunch early an undersized OLTP database will cost you customers. No body is going to order books from an online book store if the OLTP database can't update their shopping cart in less than 15 seconds.

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 proccess untill 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 programmers do in concurrent programming. Just as you'll find anywhere, when you've got a bunch of diffrent persons or proccesses all grabbing for the same thing at the same time (or at least the potential for that to occur) your going to run into problems and raw performance (getting your hands in and out as quick as possible) is generally one of the solutions.

Several other factors come into place with OLTP databases, and the Oracle10g documentation library even has a whole section dedicated just to OLTP. Find more information in the Oracle10g docs:

http://www.oracle.com/pls/db10g/portal.portal_demo3?selected=18

12.4 Data Warehousing and Data Marts

A *data warehouse* is a database dedicated to storing information log term for the purposes of reporting and decision making. It is part of a multitiered scheme of databases to provide for a variety of needs. *Data marts* effectively small data warerhouses. The diffrence is data marts tend to be departmental and fairly

application specific whereas data warehouses are more general.

An example of this multiteered scheme might be a large corperation or organization like a bank. The bank would have diffrent databses for diffrent department. The investment banking dept for instance would have a variety of databases, and the checking/savings accounts departements would have a variety of databases, and the lending departments would have their own slew of databases. Each department of the company could consolidate all their databases into a larger databases dubbed a *data mart*. Using this data mart each departement can look for trends and index/argrigate data that might otherwise be seperated or disjointed. These data marts could provide a significant ammount of operational data for decision making. But we can then go a step further and create a bigger centralized database at the corperate level dubbed an *Enterprise Data Warehouse* where we consolidate all the data marts into a giant hodge podge of data where we can do even more reporting and aggrigations to find even broader trends and statistics in order to make even better decisions.

In this general case, you can see we're building a heirarchy. The departmental database servers (probly OLTP databases) are the ones doing the real day to day work of reading and writting data for operations. But the data marts and data warehouse(s) remain primarily read-only databases. All this ties in with analytical systems like OLAP (Online Analytical Processing) and other advanced Oracle extensions like Oracle Text (discussed in the next section) to provide a powerful centralized view of your entire organization. For instance, at the data warehouse we could run daily reports to determine what trends exist between the number of daily deposites into savings and investment accounts versus the number of new mortages. Or maybe we wanted to see if there is a corolation between online banking transactions and branch transactions based on geographical location using *spatial* data.

By building an intellegent hierarchy of databases we can ensure that advanced reporting (a CPU and disk intensive task) never interfears the transaction proccessing that they need to perform to keep business moving. It can ensure that there is never a need to overextend databases just to keep everything in one place even if it doesn't need to be for typical usage. And it can ensure that you have a wholistic view of your organizations data from multiple levels whether departmental or corperate.

12.5 Oracle Options and Extensions

A wide variety of options and extensions for Oracle exist. Lets just mention breifly some of them so you at least have a basic idea of what they are.

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 Spatial databases store geographical information (GPS coordinates for instance) with records. Using Spatial you can run queries based on geographical parameters. For instance, how many of the given objects in the database are within 50 miles of New York city.

Streams Information Sharing/Distribution mechanism for replication and integration (Similar but diffrent to DataGuard)

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

Text Oracle 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, HTML/XML section searching, and so on. It can render search results in various formats including unformatted text, HTML with term highlighting, and original document format. Oracle Text supports multiple languages and uses advanced relevance-ranking technology to improve search quality. Nifty.

interMedia Enables efficent management and retrieval of images, audio and video data, including the ability to automate metadata extraction and basic image proccessing 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. The features of Personal are 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 Oracle Database 10g, including full clustering support.

Enterprise Edition Industry-leading performance, scalability, and reliability for OLTP, decision support, and content management. The whole enchilada.

12.6 Oracle Pricing

This section is obviously time sensative, but as of this writting the pricing broke down like this:

Lite Edition $100 per user.

Personal Edition $400 per user.

Standard Edition One $4,995 per CPU or $149 per user.

Standard Edition $15,000 per CPU or $300 per user.

Enterprise Edition $40,00 per CPU or $800 per user.

All of these prices are for *perpetual* licenses, meaning they are good forever. Oracle doesn't say CPU, they say "proccessor", but CPu is more fitting because it doesn't matter if you use 8 single CPU systems or 1 8-way system, it's the same cost. Because of this distiction the idea of running on small X86 systems using RAC isn't as attractive because you can get more work out of Sun UltraSparcIV or IBM POWER5 than you can out of small proccessor and therefore reducing the true total cost of your Oracle enviroment.

Obviously, the prefered way to buy Oracle is based on named users and to simply reduce the number of named users you create. Named users are defined by Oracle as:

> Named User Plus: is defined as an individual authorized by you to use the programs which are installed on a single server or multiple servers, regardless of whether the individual is actively using the programs at any given time. A non human operated device will be counted as a named user plus in addition to all individuals authorized to use the programs, if such devices can access the programs.

I called Oracle to clarify this and I was told that they consider a "named user" as any person (physical human) or resource (automated proccess) that touches the database. An in this way (so they say) you can no buy named users for a database that will be used with a webserver because each visitor to the site is a "user" of the database, even if you only have 5 webservers that actually talk directly to the database.

In addition, when buying per user, you must adhear to Oracle set "User Minimums". For instance, when purchasing Oracle Database Enterprise Edition the *Named User Plus Minimum* is 25 Named Users per Proccessor. So if you plan to install Oracle EE on an 8 way system you need to buy 200 named users, at $800 per user bringing your total cost to $160,000!

Now, on the plus side, Standard Edition has a minimum of 5 named users and it's not per processors, but the edition is limited to 4 proccessors. So you could buy Oracle Standard Edition with 5 named users for $1,500 and use it on either a 4 way box, or because RAC is included, in a 4 node RAC configuration.

Also, Oracle can sell you CD Packs, if you don't want to download all the media. The packs tend to cost less than $100. You can actually buy printed manauls, but they are $75 per manual.

To tie it all together, lets assume you were going to buy Oracle Enterprise Edition for your Sun V1280 with 12 CPUs. Thats gonna run $40,000 per CPU bringing the software cost to $480,000, 22% of that for 1 year of support translating to $105,600 (per year), $100 for the media kit, and $25 for a Polo Shirt making the grand total cost $585,725. Consider that the V1280 with 12 USIII CPUs at 1.2Ghz and 24Gb of memory only costs you $139,995.00.

12.7 License Enforcement

By this point you've most likely downloaded and installed and have been using one of the editions of Oracle (probly Enterprise Edition). You've probly noticed at some point that nothing stops you from using functionality or asks for a key or warns you that your evaluation period is almost up. Thats because there are *no* enforcement mechinisms inside of Oracle! They expect you to be honest and don't bother you with with trivialities.

Even though Oracle doesn't enforce its licenesing on your system, they sure as hell have a lot of lawyers and plenty of time to enforce licensing in the real world. Don't pull a fast one on them unless you want to put your company at serious risk. I've never heard of a company being forcably audited but dont' risk becoming the first. Besides that, the honor system is much kooler than Veritas-like lock downs.

12.8 Oracle Support

When you buy software from Oracle it *does not* come with a support agreement. Support contracts are annual and cost 22% of the purchase price. If you buy support you are entitled to phone, email, and web support (*MetaLink*), plus patches and no-cost upgrade to future releases.

Chapter 13

Tools to lessen the pain

If you've gotten this far in the book you are probly aware of two things: 1) SQL*Plus sucks, and 2) Enterprise Manager makes everything easy. Rs an SA I tend to avoid Enterprise Manager simply because I don't want to allow myself to become dependant on it. You never know when you'll need to work on a system that doesn't use EM, and I don't want to be lost because I don't have my precious EM. However, if you've used it you know that it makes Oracle feel really simple and managable. On the other side of things, SQL*Plus can give you a migrane due to it's shear stupidity. When you consider the two of them, it's hard to imagine that Oracle could put so much effort into EM but not add something as simple as Readline support into SQL*Plus. But then again, it seems like the Korn Shell is the prefrence of most DBA's so they just might not know any better.

There are two Open Source tools that will make the hurt go away and make Oracle a heck of alot easier to manage, explore and interface with: TOra and YaSQL.

13.1 YaSQL

SQL*Plus sucks. You can't really debate the point, it just does. Especially for those of us who have been spoiled with BASH and ZSH's nifty completion and history editing features it's almost unbearable to be stuck in the confines of SQL*Plus. But thanks to some thrifty coders the madness can end by using YaSQL.

YaSQL is Yet Another SQL*Plus replacement. It's a GPL app written in PERL by Jon Nangle and Nathan Shafer and is copyright Ephibian, Inc.

So what is SQL*Plus really missing? History editing is probly the big one. After typing out a long query and then screwing up a line you don't want to cut and paste the whole damned thing back in. PostgreSQL like listing would be kool 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 that i think *might* be big) it would be nice to only see the first 10 lines, or last 10 lines, or whatever instead of the whole thing, especially when you just want to see acouple rows to craft a better query. Well, good news! YaSQL does all this and *more*!

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

```
benr@nexus6 ~$ export ORACLE_HOME=/u01/app/oracle/product/10.1.0/db_1
benr@nexus6 ~$ export ORACLE_SID=cuddle
benr@nexus6 ~$ export LD_LIBRARY_PATH=$ORACLE_HOME/lib
benr@nexus6 ~$ 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
benr@cuddle>
```

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 semicolon telling YaSQL to only show me the first 5 rows of output. Then I use the "show" YaSQL command to list all the avalible tables to me (remember my rant about not having this ability like PostgreSQL?). And 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 it from within YaSQL.

Thats just a taste of the power of YaSQL. This is a professional grade interface that is easy to use, flexable, and won't leave you wondering why DBAs 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/

13.2 TOra

TOra was written by Henrik Johnson and is now owned by Quest Software. It's an open source tool that is avalible on SourceForge. UNIX versions are avaliable at no cost, but the Windows version requires a license beyond 30 days (although not inforced). 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 fo "Toolkit for Oracle". Over time, however, it has grown to also support PostgreSQL and MySQL. It has a SQL editor, table editors, a schema browser, PL/SQL editor and debugger, Security manager, a server tuning module, and much much more. I've never used the old Oracle Enterprise Manager back when it was a Win32 GUI app instead of web based, but from what I've seen TOra looks very similar to it. So if your DBAs are complaining that their old EM interface is gone and they don't like the web interface, here is your solution.

For the SA TOra makes the database extremely easy to plow around. 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. And furthermore, because it's a nice controlled GUI you can look at massive tables with thousands of rows and just scroll down instead of trying to pipe output to a file for examination or something.

The SQL and PL/SQL editors are pretty sweet. Syntax highlighting, line stepping and breakpoint debugging, tree parsing, and more. Extremely handy and much better than trial and error using just VIM and SQL*Plus.

TOra can also manage multiple connections and have a pretty nice login interface, so you can just click your instance and go. Just make sure that 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 avalible 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 website: *http://www.globecom.net/tora/*

Appendix A

Oracle Proccesses

Oracle is a big beast, to say the least. It's composed of a variety of diffrent componants. Each database is run as an *instance*. A database server can run multiple instances at a time. Each instance is made up of diffrent componants. We can see these as seperate procceses at the system level.

```
bash-2.05$ ps -ef | grep -i 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   514   329  0 15:22:24 pts/2    0:00 grep -i 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 procceses make up the instance of the database BEN running on this machine. Lets 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 later in this table) 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 datafiles. 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 DBWn processes. The database selects an appropriate default setting for this initialization parameter (or might adjust a user specified setting) based upon 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 datafiles by DBWn. This event is called a *checkpoint*. The checkpoint process is responsible for signalling DBWn at checkpoints and updating all the datafiles 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 eventually 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 Dunno.

qmnc A *queue monitor* process which monitors the message queues. Used by Oracle Streams Advanced Queuing.

mmon Performs various manageability-related background tasks.

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

j000 A job queue slave. (See cjq0)

q000 Dunno.