Expert Oracle Database Architecture

9*i* and 10*g* Programming Techniques and Solutions

Thomas Kyte

Apress®

Expert Oracle Database Architecture: 9*i* and 10*g* Programming Techniques and Solutions

Copyright © 2005 by Thomas Kyte

ISBN (pbk): 1-59059-530-0

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Tony Davis
Technical Reviewer: Jonathan Lewis, Roderick Manalac, Michael Möller, Gabe Romanescu
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis,
Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Associate Publisher: Grace Wong
Project Manager: Sofia Marchant
Copy Edit Manager: Nicole LeClerc
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Dina Quan
Proofreader: Linda Marousek
Indexer: Broccoli Information Management
Artist: Kinetic Publishing Services, LLC
Interior Designer: Van Winkle Design Group
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

CHAPTER 2

Architecture Overview

Oracle is designed to be a very portable database; it is available on every platform of relevance, from Windows to UNIX to mainframes. For this reason, the physical architecture of Oracle looks different on different operating systems. For example, on a UNIX operating system, you will see Oracle implemented as many different operating system processes, with virtually a process per major function. On UNIX, this is the correct implementation, as it works on a multiprocess foundation. On Windows, however, this architecture would be inappropriate and would not work very well (it would be slow and nonscaleable). On the Windows platform, Oracle is implemented as a single, threaded process. On IBM mainframe systems, running OS/390 and z/OS, the Oracle operating system–specific architecture exploits multiple OS/390 address spaces, all operating as a single Oracle instance. Up to 255 address spaces can be configured for a single database instance. Moreover, Oracle works together with OS/390 Workload Manager (WLM) to establish execution priority of specific Oracle workloads relative to each other and relative to all other work in the OS/390 system. Even though the physical mechanisms used to implement Oracle from platform to platform vary, the architecture is sufficiently generalized so that you can get a good understanding of how Oracle works on all platforms.

In this chapter, I present a broad picture of this architecture. We'll examine the Oracle server and define some terms such as "database" and "instance" (terms that always seem to cause confusion). We'll take a look at what happens when we "connect" to Oracle and, at a high level, how the server manages memory. In the subsequent three chapters, we'll look in detail at the three major components of the Oracle architecture:

- Chapter 3 covers *files*. In this chapter, we'll look at the set of five general categories of files that make up the database: parameter, data, temp, control, and redo log files. We'll also cover other types of files, including trace, alert, dump (DMP), data pump, and simple flat files. We'll look at the Oracle 10*g* new file area called the Flashback Recovery Area, and we'll also discuss the impact that Automatic Storage Management (ASM) has on our file storage.

- Chapter 4 covers the Oracle *memory structures* referred to as the System Global Area (SGA), Process Global Area (PGA), and User Global Area (UGA). We'll examine the relationships between these structures, and we'll also discuss the shared pool, large pool, Java pool, and various other SGA components.

- Chapter 5 covers Oracle's *physical processes or threads*. We'll look at the three different types of processes that will be running on the database: server processes, background processes, and slave processes.

It was hard to decide which of these components to cover first. The processes use the SGA, so discussing the SGA before the processes might not make sense. On the other hand, when discussing the processes and what they do, I'll need to make references to the SGA. These two components are closely tied: the files are acted on by the processes and would not make sense without first understanding what the processes do.

What I will do, then, in this chapter is define some terms and give a general overview of what Oracle looks like (if you were to draw it on a whiteboard). You'll then be ready to delve into some of the details.

Defining Database and Instance

There are two terms that, when used in an Oracle context, seem to cause a great deal of confusion: "instance" and "database." In Oracle terminology, the definitions of these terms are as follows:

- *Database*: A collection of physical operating system files or *disk*. When using Oracle 10*g* Automatic Storage Management (ASM) or RAW partitions, the database may not appear as individual separate files in the operating system, but the definition remains the same.

- *Instance*: A set of Oracle background processes/threads and a shared memory area, which is memory that is shared across those threads/processes running on a single computer. This is the place to maintain volatile, nonpersistent stuff (some of which gets flushed to disk). A database instance can exist without any disk storage whatsoever. It might not be the most useful thing in the world, but thinking about it that way will definitely help draw the line between the *instance* and the *database*.

The two terms are sometimes used interchangeably, but they embrace very different concepts. The relationship between them is that a database may be *mounted* and *opened* by many instances. An instance may *mount* and *open* a single database at any point in time. In fact, it is true to say that an instance will mount and open at most a single database in its entire lifetime! We'll look at an example of that in a moment.

Confused even more? Some further explanation should help clear up these concepts. An instance is simply a set of operating system processes, or a single process with many threads, and some memory. These processes can operate on a database; a database is just a collection of files (data files, temporary files, redo log files, and control files). At any time, an instance will have only one set of files (one *database*) associated with it. In most cases, the opposite is true as well: a database will have only one instance working on it. However, in the special case of Oracle *Real Application Clusters* (*RAC*), an option of Oracle that allows it to function on many computers in a clustered environment, we may have many instances simultaneously mounting and opening this one database, which resides on a set of shared physical disks. This gives us access to this single database from many different computers at the same time. Oracle RAC provides for extremely highly available systems and has the potential to architect extremely scalable solutions.

Let's take a look at a simple example. Say we've just installed Oracle 10*g* version 10.1.0.3. We did a software-only installation. No starter databases, nothing—just the software.

The `pwd` command shows the current working directory (this example was performed on a Linux-based computer). We're in the `dbs` directory (on Windows, this would be the `database` directory) and the `ls -l` command shows it is "empty." There is no `init.ora` file and no SPFILEs (*stored parameter files*; these will be discussed in detail in Chapter 3).

```
[ora10g@localhost dbs]$ pwd
/home/ora10g/dbs
[ora10g@localhost dbs]$ ls -l
total 0
```

Using the `ps` (process status) command, we can see all processes being run by the user `ora10g` (the Oracle software owner in this case). There are no Oracle database processes whatsoever at this point.

```
[ora10g@localhost dbs]$ ps -aef | grep ora10g
ora10g    4173  4151  0 13:33 pts/0    00:00:00 -su
ora10g    4365  4173  0 14:09 pts/0    00:00:00 ps -aef
ora10g    4366  4173  0 14:09 pts/0    00:00:00 grep ora10g
```

We then use the `ipcs` command, a UNIX command that is used to show interprocess communication devices such as shared memory, semaphores, and the like. Currently there are none in use on this system at all.

```
[ora10g@localhost dbs]$ ipcs -a

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

We then start up SQL*Plus (Oracle's command-line interface) and connect `AS SYSDBA` (the account that is allowed to do virtually anything in the database). The connection is successful and SQL*Plus reports we are connected to an idle instance:

```
[ora10g@localhost dbs]$ sqlplus "/ as sysdba"

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Dec 19 14:09:44 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
Connected to an idle instance.
SQL>
```

Our "instance" right now consists solely of the Oracle server process shown in bold in the following output. There is no shared memory allocated yet and no other processes.

```
SQL> !ps -aef | grep ora10g
ora10g    4173  4151  0 13:33 pts/0    00:00:00 -su
ora10g    4368  4173  0 14:09 pts/0    00:00:00 sqlplus   as sysdba
ora10g    4370     1  0 14:09 ?        00:00:00 oracleora10g (...)
ora10g    4380  4368  0 14:14 pts/0    00:00:00 /bin/bash -c ps -aef | grep ora10g
```

```
ora10g    4381  4380  0 14:14 pts/0    00:00:00 ps -aef
ora10g    4382  4380  0 14:14 pts/0    00:00:00 grep ora10g

SQL> !ipcs -a

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

SQL>
```

Let's try to start the instance now:

```
SQL> startup
ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file '/home/ora10g/dbs/initora10g.ora'
SQL>
```

That is the sole file that must exist in order to start up an instance—we need either a parameter file (a simple flat file described in more detail shortly) or a stored parameter file. We'll create the parameter file now and put into it the minimal information we need to actually start a database instance (normally, many more parameters will be specified, such as the database block size, control file locations, and so on):

```
$ cat initora10g.ora
db_name = ora10g
```

and then once we get back into SQL*Plus:

```
SQL> startup nomount
ORACLE instance started.
```

We used the `nomount` option to the `startup` command since we don't actually have a database to "mount" yet (see the SQL*Plus documentation for all of the startup and shutdown options).

■ **Note** On Windows, prior to running the `startup` command, you'll need to execute a service creation statement using the `oradim.exe` utility.

Now we have what I would call an "instance." The background processes needed to actually run a database are all there, such as process monitor (PMON), log writer (LGWR), and so on (these processes are covered in detail in Chapter 5).

```
Total System Global Area  113246208 bytes
Fixed Size                   777952 bytes
Variable Size              61874464 bytes
Database Buffers           50331648 bytes
Redo Buffers                 262144 bytes
SQL> !ps -aef | grep ora10g
ora10g    4173  4151  0 13:33 pts/0    00:00:00 -su
ora10g    4368  4173  0 14:09 pts/0    00:00:00 sqlplus   as sysdba
ora10g    4404     1  0 14:18 ?        00:00:00 ora_pmon_ora10g
ora10g    4406     1  0 14:18 ?        00:00:00 ora_mman_ora10g
ora10g    4408     1  0 14:18 ?        00:00:00 ora_dbw0_ora10g
ora10g    4410     1  0 14:18 ?        00:00:00 ora_lgwr_ora10g
ora10g    4412     1  0 14:18 ?        00:00:00 ora_ckpt_ora10g
ora10g    4414     1  0 14:18 ?        00:00:00 ora_smon_ora10g
ora10g    4416     1  0 14:18 ?        00:00:00 ora_reco_ora10g
ora10g    4418     1  0 14:18 ?        00:00:00 oracleora10g (...)
ora10g    4419  4368  0 14:18 pts/0    00:00:00 /bin/bash -c ps -aef | grep ora10g
ora10g    4420  4419  0 14:18 pts/0    00:00:00 ps -aef
ora10g    4421  4419  0 14:18 pts/0    00:00:00 grep ora10g
```

Additionally, ipcs is, for the first time, reporting the use of shared memory and semaphores—two important interprocess communication devices on UNIX:

```
SQL> !ipcs -a

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x99875060 458760     ora10g    660        115343360  8

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xf182650c 884736     ora10g    660        34

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

SQL>
```

Note that we have no "database" yet. We have a name of a database (in the parameter file we created), but no database whatsoever. It we tried to "mount" this database, then it would fail because it quite simply does not yet exist. Let's create it. I've been told that creating an Oracle database involves quite a few steps, but let's see:

```
SQL> create database;
Database created.
```

That is actually all there is to creating a database. In the real world, however, we would use a slightly more complicated form of the CREATE DATABASE command because we would need to tell Oracle where to put the log files, data files, control files, and so on. But here we now have a fully operational database. We would need to run the $ORACLE_HOME/rdbms/admin/catalog.sql script and other catalog scripts to build the rest of the data dictionary we use every day (the views we use such as ALL_OBJECTS are not yet present in this database), but we have a database here. We can use a simple query against some Oracle V$ views, specifically V$DATAFILE, V$LOGFILE, and V$CONTROLFILE, to list the files that make up this database:

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

NAME
--------------------------------------------------------------------------------
/home/ora10g/dbs/dbs1ora10g.dbf
/home/ora10g/dbs/dbx1ora10g.dbf

SQL> select member from v$logfile;

MEMBER
--------------------------------------------------------------------------------
/home/ora10g/dbs/log1ora10g.dbf
/home/ora10g/dbs/log2ora10g.dbf

SQL> select name from v$controlfile;

NAME
--------------------------------------------------------------------------------
/home/ora10g/dbs/cntrlora10g.dbf

SQL>
```

Oracle used defaults to put everything together and created a database as a set of persistent files. If we close this database and try to open it again, we'll discover that we can't:

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

SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-16196: database has been previously opened and closed
```

An instance can mount and open at most one database in its life. We must discard this instance and create a new one in order to open this or any other database.

To recap,

- An *instance* is a set of background processes and shared memory.
- A *database* is a collection of data stored on disk.
- An instance can mount and open only a single database, ever.
- A database may be mounted and opened by one or more instances (using RAC).

As noted earlier, there is, in most cases, a one-to-one relationship between an instance and a database. This is probably how the confusion surrounding the terms arises. In most peoples' experience, a database is an instance, and an instance is a database.

In many test environments, however, this is not the case. On my disk, I might have five separate databases. On the test machine, at any point in time there is only one instance of Oracle running, but the database it is accessing may be different from day to day or hour to hour, depending on my needs. By simply having many different configuration files, I can mount and open any one of these databases. Here, I have one "instance" at a time but many databases, only one of which is accessible at any point in time.

So now when someone talks about an instance, you'll know they mean the processes and memory of Oracle. When they mention the database, they are talking about the physical files that hold the data. A database may be accessible from many instances, but an instance will provide access to exactly one database at a time.

The SGA and Background Processes

You're probably ready now for an abstract picture of what an Oracle instance and database looks like (see Figure 2-1).

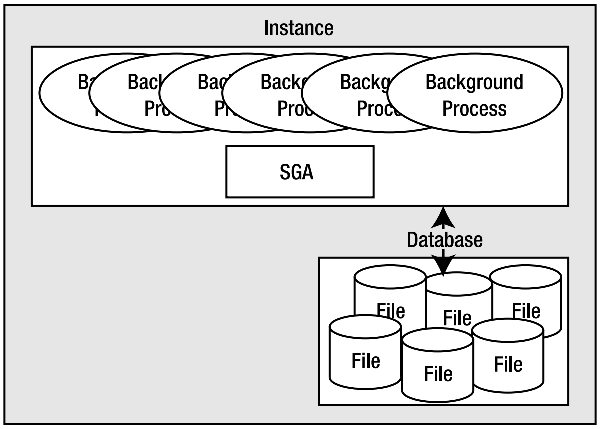

Figure 2-1. *Oracle instance and database*

Figure 2-1 is a depiction of an Oracle instance and database in their simplest form. Oracle has a large chunk of memory called the SGA where it will, for example, do the following:

- Maintain many internal data structures that all processes need access to.
- Cache data from disk; buffer redo data before writing it to disk.
- Hold parsed SQL plans.
- And so on.

Oracle has a set of processes that are "attached" to this SGA, and the mechanism by which they attach differs by operating system. In a UNIX environment, they will physically attach to a large *shared memory* segment, a chunk of memory allocated in the operating system that may be accessed by many processes concurrently (generally using `shmget()` and `shmat()`).

Under Windows, these processes simply use the C call `malloc()` to allocate the memory, since they are really threads in one big process and hence share the same virtual memory space. Oracle will also have a set of files that the database processes/threads read and write (and Oracle processes are the only ones allowed to read or write these files). These files hold all of our table data, indexes, temporary space, redo logs, and so on.

If you were to start up Oracle on a UNIX-based system and execute a `ps` command, you would see that many physical processes are running, with various names. You saw an example of that earlier when you observed the `pmon`, `smon`, and other processes. I cover what each of these processes are in Chapter 5, so just be aware for now that they are commonly referred to as the *Oracle background processes*. They are persistent processes that make up the instance, and you will see them from the time you start the instance until the time you shut it down.

It is interesting to note that these are processes, not individual programs. There is only one Oracle binary executable on UNIX; it has many "personalities" depending on what it was told to do when it starts up. The same binary executable that was run to get `ora_pmon_ora10g` was also used to get the process `ora_ckpt_ora10g`. There is only one binary executable program, named simply `oracle`. It is just executed many times with different names.

On Windows, using the `pslist` tool (`http://www.sysinternals.com/ntw2k/freeware/pslist.shtml`), we'll find only one process, `oracle.exe`. Again, on Windows there is only one binary executable (`oracle.exe`). Within this process, we'll find many threads representing the Oracle background processes.

Using `pslist` (or any of a number of tools), we can see these threads:

```
C:\Documents and Settings\tkyte>pslist oracle

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for ORACLE-N15577HE:

Name                Pid Pri Thd  Hnd   Priv        CPU Time    Elapsed Time
oracle             1664   8  19  284 354684     0:00:05.687     0:02:42.218
```

Here we can see there are 19 threads (`Thd` in the display) contained in the single Oracle process. These threads represent what were processes on UNIX—they are the `pmon`, `arch`, `lgwr`, and so on bits of Oracle. We can use `pslist` to see more details about each:

```
C:\Documents and Settings\tkyte>pslist -d oracle

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Thread detail for ORACLE-N15577HE:

oracle 1664:
```

```
Tid Pri    Cswtch            State     User Time   Kernel Time   Elapsed Time
1724   9       148   Wait:Executive  0:00:00.000   0:00:00.218    0:02:46.625
 756   9       236     Wait:UserReq  0:00:00.000   0:00:00.046    0:02:45.984
1880   8         2     Wait:UserReq  0:00:00.000   0:00:00.000    0:02:45.953
1488   8       403     Wait:UserReq  0:00:00.000   0:00:00.109    0:02:10.593
1512   8       149     Wait:UserReq  0:00:00.000   0:00:00.046    0:02:09.171
1264   8       254     Wait:UserReq  0:00:00.000   0:00:00.062    0:02:09.140
 960   9       425     Wait:UserReq  0:00:00.000   0:00:00.125    0:02:09.078
2008   9       341     Wait:UserReq  0:00:00.000   0:00:00.093    0:02:09.062
1504   8      1176     Wait:UserReq  0:00:00.046   0:00:00.218    0:02:09.015
1464   8        97     Wait:UserReq  0:00:00.000   0:00:00.031    0:02:09.000
1420   8       171     Wait:UserReq  0:00:00.015   0:00:00.093    0:02:08.984
1588   8       131     Wait:UserReq  0:00:00.000   0:00:00.046    0:02:08.890
1600   8        61     Wait:UserReq  0:00:00.000   0:00:00.046    0:02:08.796
1608   9         5       Wait:Queue  0:00:00.000   0:00:00.000    0:02:01.953
2080   8        84     Wait:UserReq  0:00:00.015   0:00:00.046    0:01:33.468
2088   8       127     Wait:UserReq  0:00:00.000   0:00:00.046    0:01:15.968
2092   8       110     Wait:UserReq  0:00:00.000   0:00:00.015    0:01:14.687
2144   8       115     Wait:UserReq  0:00:00.015   0:00:00.171    0:01:12.421
2148   9       803     Wait:UserReq  0:00:00.093   0:00:00.859    0:01:09.718
```

We cannot see the thread "names" like we could on UNIX (`ora_pmon_ora10g` and so on) but we can see the thread IDs (`Tid`), priorities (`Pri`), and other operating system accounting information about them.

Connecting to Oracle

In this section, we'll take a look at the mechanics behind the two most common ways to have requests serviced by an Oracle server: *dedicated server* and *shared server* connections. We'll see what happens on the client and the server in order to establish connections, so we can log in and actually do work in the database. Lastly, we'll take a brief look at how to establish TCP/IP connections—*TCP/IP* being the primary networking protocol used to connect over the network to Oracle—and at how the *listener* process on our server, which is responsible for establishing the physical connection to the server, works differently in the cases of dedicated and shared server connections.

Dedicated Server

Figure 2-1 and the `pslist` output presented a picture of what Oracle looks like immediately after starting. If we were now to log into this database using a dedicated server, we would see a new process get created just to service us:

```
C:\Documents and Settings\tkyte>sqlplus tkyte/tkyte

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Dec 19 15:41:53 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
Connected to:
```

```
Oracle Database 10g Enterprise Edition Release 10.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options

tkyte@ORA10G> host pslist oracle

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for ORACLE-N15577HE:

Name                Pid Pri Thd  Hnd   Priv        CPU Time    Elapsed Time
oracle             1664   8  20  297 356020     0:00:05.906     0:03:21.546
tkyte@ORA10G>
```

Now you can see there are 20 threads instead of 19, the extra thread being our dedicated server process (more information on what exactly a dedicated server process is shortly). When we log out, the extra thread will go away. On UNIX, we would see another process get added to the list of Oracle processes running, and that would be our dedicated server.

This brings us to the next iteration of the previous diagram. Now, if we were to connect to Oracle in its most commonly used configuration, we would see something like Figure 2-2.

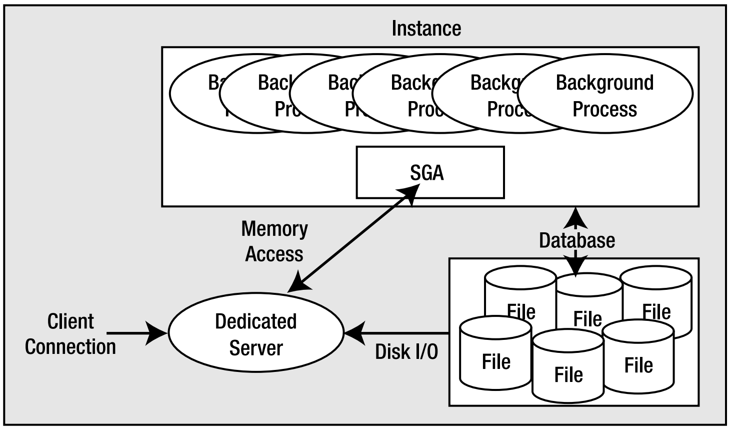

Figure 2-2. *Typical dedicated server configuration*

As noted, typically Oracle will create a new process for me when I log in. This is commonly referred to as the *dedicated server* configuration, since a server process will be dedicated to me for the life of my session. For each session, a new dedicated server will appear in a one-to-one mapping. This dedicated server process is not (by definition) part of the instance. My client process (whatever program is trying to connect to the database) will be in direct communication with this dedicated server over some networking conduit, such as a TCP/IP socket. It is this server process that will receive my SQL and execute it for me. It will read data files if necessary, and it will look in the database's cache for my data. It will perform my update statements. It will run my PL/SQL code. Its only goal is to respond to the SQL calls that I submit to it.

Shared Server

Oracle may also accept connections in a manner called *shared server* (formally known as Multi-Threaded Server, or MTS), in which we would *not* see an additional thread created or a new UNIX process appear for each user connection. In shared server, Oracle uses a pool of "shared processes" for a large community of users. Shared servers are simply a connection pooling mechanism. Instead of having 10,000 dedicated servers (that's a lot of processes or threads) for 10,000 database sessions, shared server allows us to have a small percentage of this number of processes/threads, which are (as the name implies) shared by all sessions. This allows Oracle to connect many more users to the database than would otherwise be possible. Our machine might crumble under the load of managing 10,000 processes, but managing 100 or 1,000 processes is doable. In shared server mode, the shared processes are generally started up with the database and just appear in the `ps` list.

A big difference between shared and dedicated server connections is that the client process connected to the database never talks directly to a shared server, as it would to a dedicated server. It cannot talk to a shared server because that process is, in fact, shared. To share these processes, we need another mechanism through which to "talk." Oracle employs a process (or set of processes) called *dispatchers* for this purpose. The client process will talk to a dispatcher process over the network. The dispatcher process will put the client's request into the request queue in the SGA (one of the many things the SGA is used for). The first shared server that is not busy will pick up this request and process it (e.g., the request could be `UPDATE T SET X = X+5 WHERE Y = 2`). Upon completion of this command, the shared server will place the response in the invoking dispatcher's response queue. The dispatcher process is monitoring this queue and, upon seeing a result, will transmit it to the client. Conceptually, the flow of a shared server request looks like Figure 2-3.

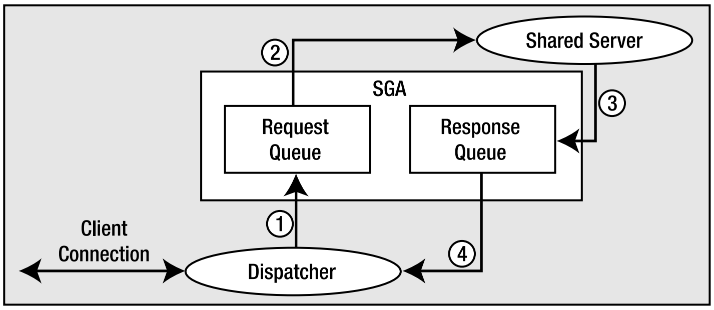

Figure 2-3. *Steps in a shared server request*

As shown in Figure 2-3, the client connection will send a request to the dispatcher. The dispatcher will first place this request onto the request queue in the SGA (1). The first available shared server will dequeue this request (2) and process it. When the shared server completes, the response (return codes, data, and so on) is placed into the response queue (3), subsequently picked up by the dispatcher (4), and transmitted back to the client.

As far as the developer is concerned, there is no difference between a shared server connection and a dedicated server connection.

Now that you understand what dedicated server and shared server connections are, you may have the following questions:

- How do I get connected in the first place?
- What would start this dedicated server?
- How might I get in touch with a dispatcher?

The answers depend on your specific platform, but the sections that follow outline the process in general terms.

Mechanics of Connecting over TCP/IP

We'll investigate the most common networking case: a network-based connection request over a TCP/IP connection. In this case, the client is situated on one machine and the server resides on another machine, with the two connected on a TCP/IP network. It all starts with the client. The client makes a request using the Oracle client software (a set of provided *application program interfaces*, or *APIs*) to connect to database. For example, the client issues the following:

```
[tkyte@localhost tkyte]$ sqlplus scott/tiger@ora10g.localdomain

SQL*Plus: Release 10.1.0.3.0 - Production on Sun Dec 19 16:16:41 2004
Copyright (c) 1982, 2004, Oracle.  All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options

scott@ORA10G>
```

Here, the client is the program SQL*Plus, `scott/tiger` is the username/password, and `ora10g.localdomain` is a TNS service name. *TNS* stands for *Transparent Network Substrate* and is "foundation" software built into the Oracle client that handles our remote connections, allowing for peer-to-peer communication. The TNS connection string tells the Oracle software how to connect to the remote database. Generally, the client software running on your machine will read a file called `tnsnames.ora`. This is a plain-text configuration file commonly found in the `[ORACLE_HOME]\network\admin` directory (`[ORACLE_HOME]` represents the full path to your Oracle installation directory). It will have entries that look like this:

```
ORA10G.LOCALDOMAIN =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost.localdomain)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = ora10g)
    )
  )
```

This configuration information allows the Oracle client software to map the TNS connection string we used, `ora10g.localdomain`, into something useful—namely, a hostname, a port on that host on which a "listener" process will accept connections, the *service name* of the database on the host to which we wish to connect, and so on. A service name represents groups of applications with common attributes, service level thresholds, and priorities. The number of instances offering the service is transparent to the application, and each database instance may register with the listener as willing to provide many services. So, services are mapped to physical database instances and allow the DBA to associate certain thresholds and priorities with them.

This string, `ora10g.localdomain`, could have been resolved in other ways. For example, it could have been resolved using Oracle Internet Directory (OID), which is a distributed Lightweight Directory Access Protocol (LDAP) server, similar in purpose to DNS for hostname resolution. However, use of the `tnsnames.ora` file is common in most small to medium installations where the number of copies of such a configuration file is manageable.

Now that the client software knows where to connect to, it will open a TCP/IP socket connection to the server with the hostname `localhost.localdomain` on port 1521. If the DBA for our server has installed and configured Oracle Net, and has the listener listening on port 1521 for connection requests, this connection may be accepted. In a network environment, we will be running a process called the *TNS listener* on our server. This listener process is what will get us physically connected to our database. When it receives the inbound connection request, it inspects the request and, using its own configuration files, either rejects the request (e.g., because there is no such database, or perhaps our IP address has been disallowed connections to this host) or accepts it and goes about getting us connected.

If we are making a dedicated server connection, the listener process will create a dedicated server for us. On UNIX, this is achieved via a `fork()` and `exec()` system call (the only way to create a new process after initialization in UNIX is via `fork()`). The new dedicated server process inherits the connection established by the listener, and we are now physically connected to the database. On Windows, the listener process requests the database process to create a new thread for a connection. Once this thread is created, the client is "redirected" to it, and we are physically connected. Diagrammatically in UNIX, it would look as shown in Figure 2-4.

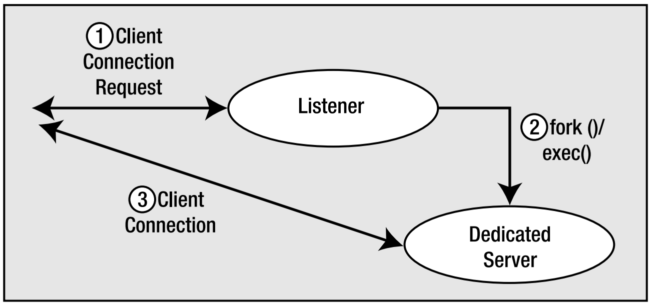

Figure 2-4. *The listener process and dedicated server connections*

On the other hand, the listener will behave differently if we are making a shared server connection request. This listener process knows the dispatcher(s) we have running in the instance. As connection requests are received, the listener will choose a dispatcher process from the pool of available dispatchers. The listener will either send back to the client the connection information describing how the client can connect to the dispatcher process or, if possible, "hand off" the connection to the dispatcher process (this is operating system– and database version–dependent, but the net effect is the same). When the listener sends back the connection information, it is done because the listener is running on a well-known hostname and port on that host, but the dispatchers will be accepting connections on randomly assigned ports on that server. The listener is made aware of these random port assignments by the dispatcher and will pick a dispatcher for us. The client then disconnects from the listener and connects directly to the dispatcher. We now have a physical connection to the database. Figure 2-5 illustrates this process.

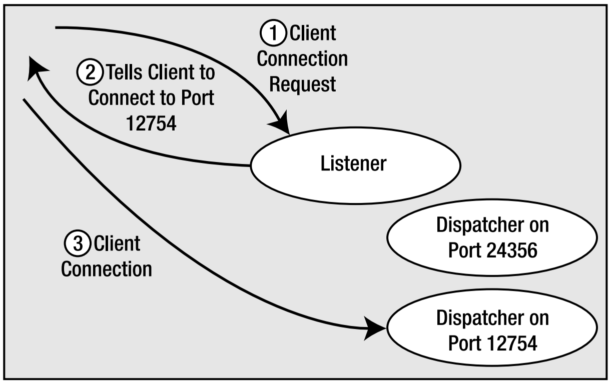

Figure 2-5. *The listener process and shared server connections*

Summary

That completes our overview of the Oracle architecture. In this chapter, we defined the terms "instance" and "database" and saw how to connect to the database through either a dedicated server connection or a shared server connection. Figure 2-6 sums up the material covered in the chapter and shows the interaction between a client using a shared server connection and a client using a dedicated server connection. It also shows that an Oracle instance may use both connection types simultaneously. (In fact, an Oracle database *always* supports dedicated server connections—even when configured for shared server.)

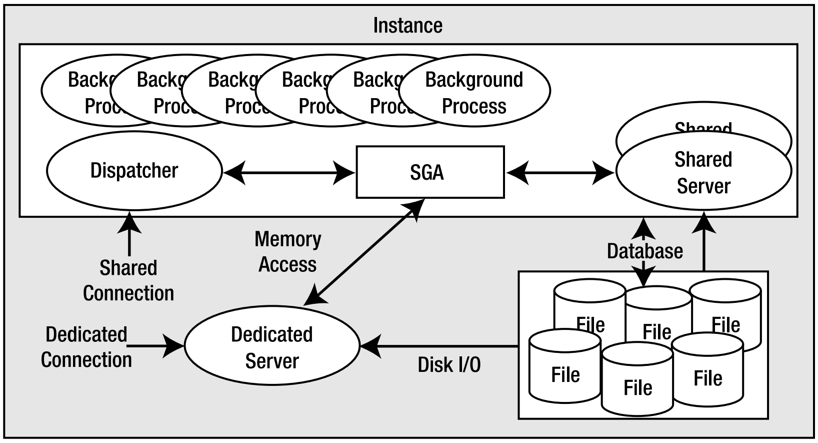

Figure 2-6. *Connection overview*

Now you're ready to take a more in-depth look at the processes behind the server, what they do, and how they interact with each other. You're also ready to look inside the SGA to see what it contains and what its purpose is. You'll start in the next chapter by looking at the types of files Oracle uses to manage the data and the role of each file type.