CHAPTER 3

Oracle8i Architecture

In This Chapter

Understanding the differences between an instance and a database

Examining database file architecture

Understanding memory architecture

Looking at process architecture

Architecture refers to the way in which all the pieces of an Oracle8i database instance were designed to work together. It encompasses the way in which Oracle8i uses memory, the way in which Oracle8i uses disk files, and the way in which various Oracle8i processes interact with each other.

It's important for you to understand Oracle8i's architecture, particularly from a tuning standpoint, because you can't hope to properly tune an Oracle8i database if you don't know something about how Oracle operates. Understanding the architecture also gives you a lot of insight into some of the terminology that you'll encounter, and it also helps you to appreciate why things work the way that they do. For example, you'll understand why you have to mount a database before you can rename a database file. This chapter opens with a discussion of the fundamental differences between an instance and a database, and then it covers database files, memory, and process architectures.

Understanding the Differences Between Instance and Database

Instance? Database? You'll run into these two words often as a DBA. They are simple but important words because the entire Oracle8i architecture is derived from them. Chapter 2, "Oracle8i Overview," describes the difference between an instance and a database. Let's recap that discussion here.

A *database*, in the Oracle world, refers to a collection of files used to store and manage related data. The term database refers only to the files, nothing else. Of course, if you have a

database full of information, you probably want to do something with that information. That's where the Oracle instance comes into play.

An *instance* is a set of processes that work together to operate on your database. For performance reasons, because these processes work so closely together, they share access to an area of memory known as the *system global area* (*SGA*). The SGA is also considered to be part of the instance.

The processes that make up an Oracle instance allow you to change and retrieve your data. Some of the processes also work to protect the integrity of your data and ensure the recoverability of your data in the event of a system crash, loss of a disk, or other unforeseen event. Most Oracle processes are referred to as *background processes* because they are always running and they aren't associated with any particular user.

When an Oracle instance is running and a database is being used, the interaction between the various processes, database files, and shared memory will look something like the diagram in Figure 3-1.

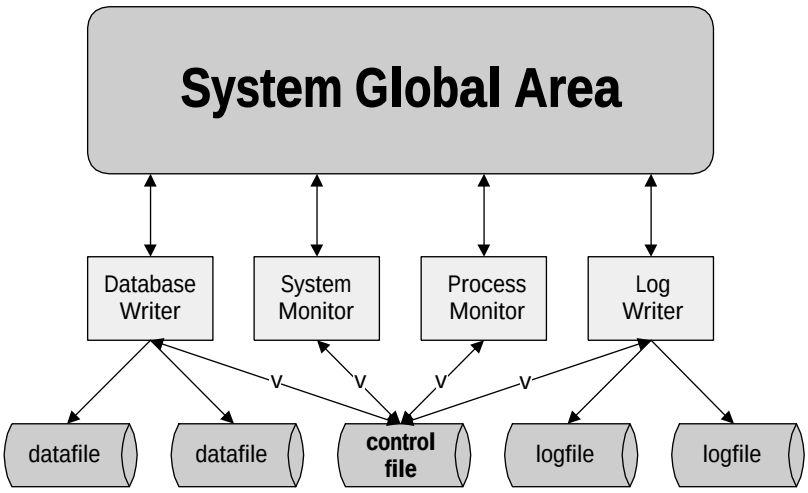


Figure 3-1: A typical Oracle instance

The configuration shown in Figure 3-1 is known as a *stand-alone instance* configuration. One instance is operating on one database. Sites that must service a lot of users and that require a high degree of availability may end up using Oracle Parallel Server. Oracle Parallel Server allows you to have many instances, all on different machines, operating on one Oracle database. Figure 3-2 shows a Parallel Server configuration.

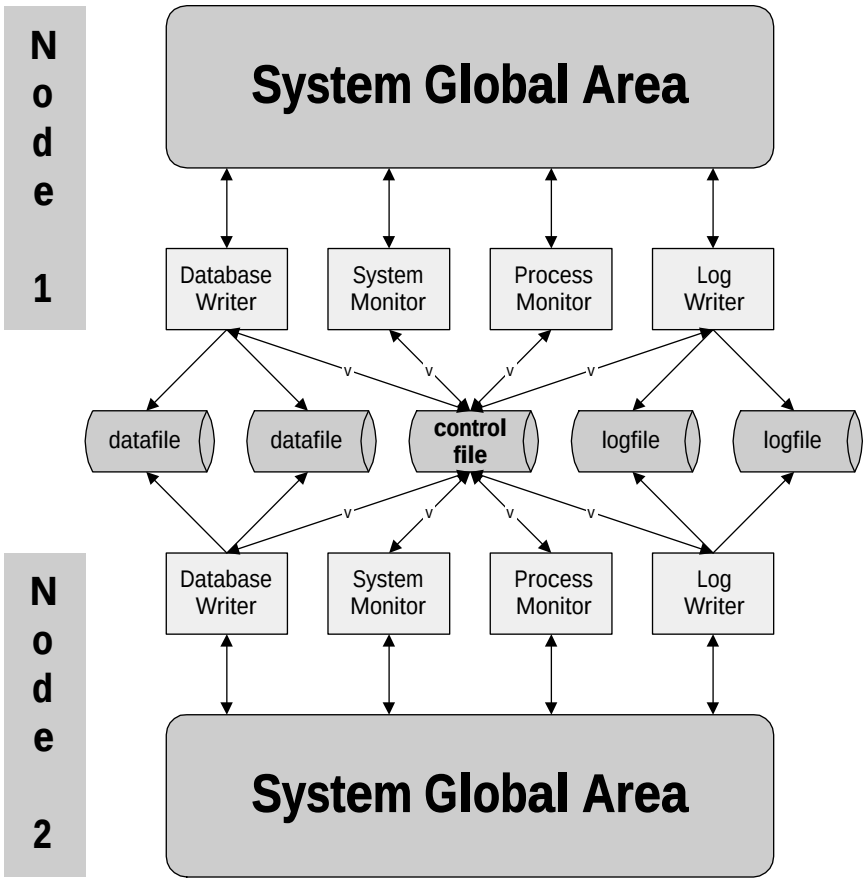


Figure 3-2: An Oracle Parallel Server configuration

Oracle Parallel Server is available only with the Enterprise Edition of Oracle.

Because Oracle Parallel Server allows many instances to open a database, you are somewhat insulated from problems that might occur if an instance crashes. In the event of a crash, the other instances will continue to run. Users will still be able to connect and get work done. In a stand-alone configuration, if an instance crashes, everyone is dead in the water until you can restart it.

You can also use Oracle Parallel Server to increase your database's throughput because the load can be spread over several computers instead of just one. This works best if you can partition your database tables and indexes in a way that minimizes the number of times that two instances will require access to the same data.

Examining Database File Architecture

Several different types of files combine to make up an Oracle8i database. This section describes each type. You'll learn the purpose of each type of file, see what type of information files contain, and learn how to locate the files that make up your database. Refer to Figure 3-3, which shows the files that you will see in an Oracle8i database.

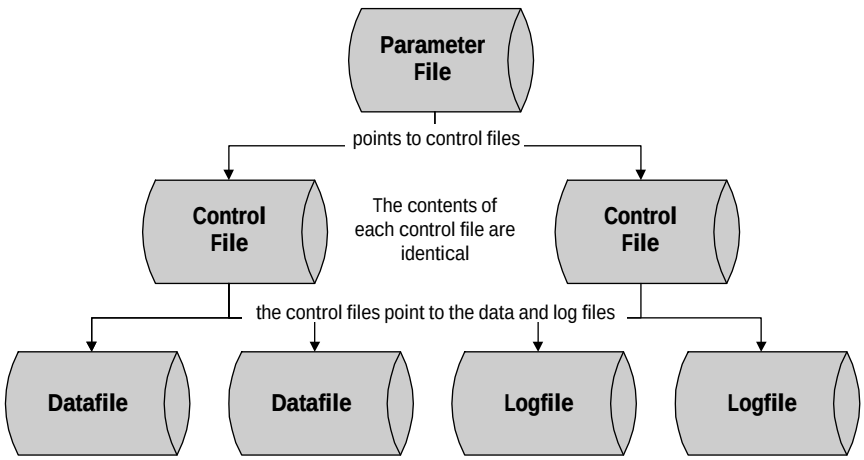


Figure 3-3: Database file types

Figure 3-3 shows the parameter file as a database file. Strictly speaking, that's not correct. However, understanding some points about the parameter file will help you better understand how Oracle keeps track of control files and archived log files. The next few sections discuss in detail each of the file types shown in Figure 3-3.

Using parameter files

The parameter file is read by either SQL*Plus, Server Manager, or Instance Manager when you use one of those programs to start an instance. It's not opened by the instance; consequently, it isn't considered to be a database file.

Even though the parameter file isn't a database file, it is important. Without the correct parameter file, you may not be able to open your database. Even if you did manage to open a database using the wrong parameter file, it might not work the way you expect because the parameter file settings greatly influence how an Oracle instance functions. You will want to treat your parameter files with the same care as you do your database files.

Parameter File Contents

Parameter files are text files. You can use any text editor to open them. If you are running UNIX, you can use the `vi` editor. On Windows NT, you can use Notepad. Parameter files serve much the same purpose as Windows .ini files. They contain a number of settings that influence how an Oracle database instance functions. Some of the more important aspects that you can control via settings in the parameter file are the following:

- The location of the database control files
- The amount of memory Oracle uses to buffer data that has been read from disk
- The amount of memory Oracle uses to buffer SQL statement execution plans, PL/SQL procedures, and data dictionary information so that they don't have to be continuously read from disk
- The default optimizer choice

Relative to database files, the parameter file performs two important functions. One is to point to the control files for a database. The other is to point to the archive log destination for a database. If you were to open the parameter file for an Oracle database, you would see lines like this:

```
control_files = ("E:\Oracle\oradata\jonathan\control01.ctl",
                 "F:\Oracle\oradata\jonathan\control02.ctl")
log_archive_dest_1 =
                "location=G:\Oracle\oradata\jonathan\archive"
```

The `control_files` entry shown here tells Oracle where to find the control files for a database. Once an instance has found the control files, it can open those files and read the locations of all the other database files. The second entry, for `log_archive_dest_1`, tells Oracle where it should copy redo log files as they are filled.

Location of Parameter Files

You can place parameter files anywhere you like. However, it's usually easiest to follow some well-established conventions about where Oracle expects these files to be and how Oracle expects them to be named.

Parameter file locations on UNIX systems

On most UNIX systems, Oracle expects database parameter files to be in the following directory:

```
$ORACLE_HOME/dbs
```

The naming convention used for parameter files on UNIX systems is initXXXX.ora, where "XXXX" represents the instance name. So if you have a database instance

named ORCL, which is the default name of Oracle's starter database, your parameter file would be named initORCL.ora.

Parameter file locations on Windows NT systems

On Windows NT systems, for whatever reason, Oracle expects parameter files to be in the database directory instead of the dbs directory. If you've performed a default install of Oracle on Windows NT, the full path to the database directory will look like this:

```
C:\Oracle\Ora8i\Database
```

Not only is the directory path for parameter files different under Windows NT, but the naming convention is different as well. For a database instance named ORCL, the Windows NT version of Oracle expects the parameter file to be named initORCL.ora.

Using control files

Oracle uses control files to store information about the state of your database. In a way, you could look at a control file as sort of a scratchpad. Whenever Oracle needs to save an important tidbit of information about the database, it writes it on the scratchpad. That's not to say that the information in a control file is unimportant. In fact, it's so important, and so critical, that Oracle recommends that you always mirror your control files — maintaining two or three copies so that if one is lost, you have the others to fall back on.

Control File Contents

Database control files contain the following major types of information:

- ✦ The database name
- ✦ Information about tablespaces
- ✦ The names and locations of all the datafiles
- ✦ The names and locations of all the redo log files
- ✦ The current log sequence number
- ✦ Checkpoint information
- ✦ Information about redo logs and the current state of archiving

The control file leads Oracle to the rest of the database files. When you start an instance, Oracle reads the control file names and locations from the parameter file. When you mount a database, Oracle opens the control file. When you finally open a database, Oracle reads the list of database files from the control file and opens each of them.

Will the Real Parameter File Please Stand Up?

Chances are, if you've installed a recent release of Oracle, your parameter files aren't where they appear to be. At least, the real parameter files won't be in either the dbs or the database directory. Instead, you will find them underneath the $ORACLE_BASE/admin/XXXX/pfile directory (UNIX) or C:\Oracle\Admin\XXXX\pfile (Windows NT), where XXXX represents the instance name.

Why is this? It has to do with a set of file-naming and placement guidelines that Oracle developed known as the optimal flexible architecture (OFA). The OFA guidelines state that the Oracle home directory should be used only for Oracle software distribution and not for any administrative or database files. This means that parameter files should not be in either the dbs or the database directories.

Rather than change the expected locations of the parameter files in its software, Oracle implemented the guidelines by taking advantage of an operating system feature known as a *file system link*. Links allow you to create directory entries that make it look like a file is in one place, when in fact it is really somewhere else.

On UNIX systems, the parameter files in the dbs directory are often nothing more than links to the real parameter files somewhere else. Windows NT doesn't support file system links, so Windows NT parameter files in the database directory often contain just one line—an include directive that points to the real parameter file somewhere else.

Finding Your Control Files

You can find the names and locations of the control files for a database in two ways. One is to look in the parameter file and find the `control_files` entry. That entry looks like this:

```
control_files = ("E:\Oracle\oradata\jonathan\control01.ctl",
                 "F:\Oracle\oradata\jonathan\control02.ctl")
```

The `control_file` entry will list all the control files that were opened when the database was started. Each control file contains identical information. If one is lost, Oracle can continue by using the other.

A second way to find the control files for a database is to log on as the `SYSTEM` user, or some other privileged user, and issue the following `SELECT` statement:

```
SELECT * FROM v$controlfile
```

You can issue this query from SQL*Plus, as shown in this screen output example:

```
SQL> SELECT * FROM v$controlfile;

STATUS  NAME
------- ----------------------------------------
        E:\ORACLE\ORADATA\JONATHAN\CONTROL01.CTL
        F:\ORACLE\ORADATA\JONATHAN\CONTROL02.CTL
```

Of course, for this second method to work, the instance must be running.

Using datafiles

Not surprisingly, Oracle stores your data in datafiles. Datafiles also typically represent the bulk of an Oracle database in terms of the disk space that they use. In terms of quantity also, you will probably have more datafiles than any other type of file.

Datafile Contents

Datafiles contain the following types of data:

- ✦ Table data
- ✦ Index data
- ✦ Data dictionary definitions
- ✦ Information necessary to undo transactions (rollback data)
- ✦ Code for stored procedures, functions, and packages
- ✦ Temporary data, often used for sorting

For performance reasons, it's usually best to separate data by type, each into its own file or set of files, and place those files on separate disks. This is especially true for data dictionary information, rollback data, and temporary data. You will almost universally store these types of data separately from the others.

Finding Your Datafiles

To generate a list of the datafiles that constitute your database, you can query a dynamic performance view named `V$DATAFILE`. The following query will give you the status, the size, and the name for each datafile in your database:

```
SELECT status, bytes, name
FROM v$datafile
```

The following example shows the results you will get if you execute this query from SQL*Plus. The two `COLUMN` commands format the output of the bytes and name the columns to make the output more readable.

```
SQL> COLUMN name FORMAT A40
SQL> COLUMN bytes FORMAT 999,999,999
SQL> SELECT status, bytes, name
  2  FROM v$datafile;

STATUS         BYTES NAME
------- ------------ ----------------------------------------
SYSTEM   167,772,160 E:\ORACLE\ORADATA\JONATHAN\SYSTEM01.DBF
ONLINE     3,145,728 E:\ORACLE\ORADATA\JONATHAN\USERS01.DBF
ONLINE    26,214,400 E:\ORACLE\ORADATA\JONATHAN\RBS01.DBF
ONLINE     2,097,152 E:\ORACLE\ORADATA\JONATHAN\TEMP01.DBF
ONLINE     5,242,880 E:\ORACLE\ORADATA\JONATHAN\OEMREP01.DBF
ONLINE     2,097,152 E:\ORACLE\ORADATA\JONATHAN\INDX01.DBF
ONLINE    31,457,280 E:\ORACLE\ORADATA\JONATHAN\USERS02.DBF

7 rows selected.
```

The status column tells you whether Oracle has the file open. A status of `ONLINE` or `SYSTEM` means that Oracle has the file open and that the data within that file is accessible. A status of `OFFLINE` means that the file is closed. Files may be offline because they were taken offline purposely by the DBA or because of a problem, such as a drive failure, that makes the file inaccessible to Oracle. You cannot access the data in an offline datafile.

The `SYSTEM` status indicates that a file is part of the system tablespace. The system tablespace is the one that contains the data dictionary. You'll learn more about tablespaces in Chapter 6, "Database Space Management." Oracle needs the data dictionary to access the data in the other datafiles, so the system tablespace always has to be open whenever the database is open.

Using log files

Log files, sometimes called redo log files, are an important component of any database. They exist to ensure the recoverability of a database in the event of a system crash, a drive failure, or any other unforeseen interruption to normal operations.

There are two types of redo log files: online redo log files and archived redo log files. Archived redo log files are sometimes referred to as offline redo log files. All Oracle databases use online redo log files. Archived redo log files are only generated when a database is run in archivelog mode, and allow for up-to-the-minute recovery in the event that a datafile is lost.

Log File Contents

Log files contain a sequential record of changes to a database. Any time you execute a SQL statement that changes the data in your database, Oracle generates one or more redo log entries to record that change. If you just change one row in a

table, one redo log entry might be enough to document that change. If you create an index on a large table, Oracle generates a prodigious number of redo log entries to describe the changes being made.

To protect the redo log from being lost, Oracle writes it as fast as possible to a set of disk files known as redo log files. When you commit a transaction, Oracle waits until all redo log entries for that transaction have been written to the log files before telling you that the commit was successful. That way, you can't possibly lose any committed changes. Each Oracle database contains at least two redo log files, and often more. Oracle writes to these files in a circular fashion, as shown in Figure 3-4.

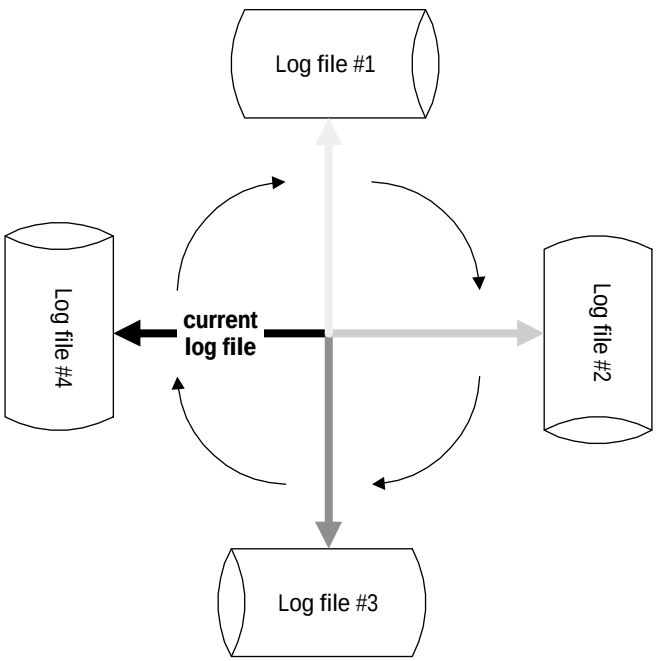


Figure 3-4: Oracle writes to the redo log files in a circular fashion.

As with control files, redo log files should always be mirrored. You can use hardware mirroring if your system is capable of it, or you can have the Oracle software mirror the files for you.

In a production setting, you should keep all redo log files that have been created since the most recent full backup of the database. Do this by having Oracle copy each redo log file as it is filled to a secure, long-term storage location. This process is known as *archiving* a redo log file. This gets to the heart of what the redo log buys

you. Beginning with the most recent full backup of your database, the redo log files (both archived and online) provide you with a history of all changes made up to the current moment.

When you lose a disk drive, and someday you will, you'll lose all the datafiles on that drive. The redo log allows you to recover from what would otherwise be a disastrous event by simply performing the following three steps:

1. Replace the disk with a new one that works.
2. Restore the lost files from the most recent backup.
3. Use the redo log to reapply the changes to the restored files, bringing them up to date.

There are alternatives to replacing the disk. You could just as easily restore the files to a different disk and use the `ALTER DATABASE RENAME FILE` command to tell Oracle about the new location.

Oracle automates the third step. If you've configured your system correctly, the process of reapplying changes from the redo log is painless. All you have to do is issue one short command, sit back, and watch.

Finding Your Log Files

To generate a list of the online log files in your database, query the `V$LOGFILE` view. You need to be logged in as SYSTEM, or some other privileged user, to do this. The following query returns the status and name of all your online redo log files:

```
SELECT member
FROM v$logfile
```

You can execute this query from SQL*Plus, as shown in this example:

```
SQL> COLUMN member FORMAT A40
SQL> SELECT member FROM v$logfile;

MEMBER
----------------------------------------
E:\ORACLE\ORADATA\JONATHAN\REDO04.LOG
E:\ORACLE\ORADATA\JONATHAN\REDO03.LOG
E:\ORACLE\ORADATA\JONATHAN\REDO02.LOG
E:\ORACLE\ORADATA\JONATHAN\REDO01.LOG
```

Finding your offline, or archived, log files is a different matter. Technically, they aren't considered part of the database. However, Oracle does keep track of them. You can query the v$archived_log view to get a list. You can find out the name of the directory, or device, to which Oracle is copying the archived log files by issuing

an ARCHIVE LOG LIST command. Consider this example showing how to query the v$archived_log view:

```
SQL> SELECT name
  2  FROM v$archived_log
  3  ORDER BY recid;

NAME
--------------------------------
D:\ORADATA\JONATHAN\ARCH_299.1
D:\ORADATA\JONATHAN\ARCH_300.1
D:\ORADATA\JONATHAN\ARCH_301.1
D:\ORADATA\JONATHAN\ARCH_302.1
D:\ORADATA\JONATHAN\ARCH_303.1
```

To use the ARCHIVE LOG LIST command, you have to connect as either SYSDBA or SYSOPER, and issue the ARCHIVE LOG LIST command. Consider this example:

```
SQL> CONNECT system/manager@jonathan.gennick as SYSDBA;
Connected.
SQL> ARCHIVE LOG LIST
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            G:\Oracle\Ora81\RDBMS
Oldest online log sequence     138
Current log sequence           141
```

Here, the archive destination is G:\Oracle\Ora81\RDBMS. As each redo log file is archived, it is copied to that directory. Knowing that, you can easily go to that directory, issue the dir command, and see a list of all your archived log files.

The archive destination that you see with the ARCHIVE LOG LIST command is only the current destination. You can change this destination dynamically, but Oracle won't keep track of your changes. It only knows about the current destination.

Understanding Memory Architecture

When an Oracle instance starts up, it allocates a large block of memory known as the SGA. The SGA for an instance is shared by all of the background processes for that instance. In addition, each process associated with an instance will have its own private area of memory known as a program global area (PGA). Figure 3-5 illustrates this.

Sharing memory this way allows the background processes to communicate with each other quickly and minimizes the overhead of interprocess communication. This contributes greatly to Oracle's efficiency as a database.

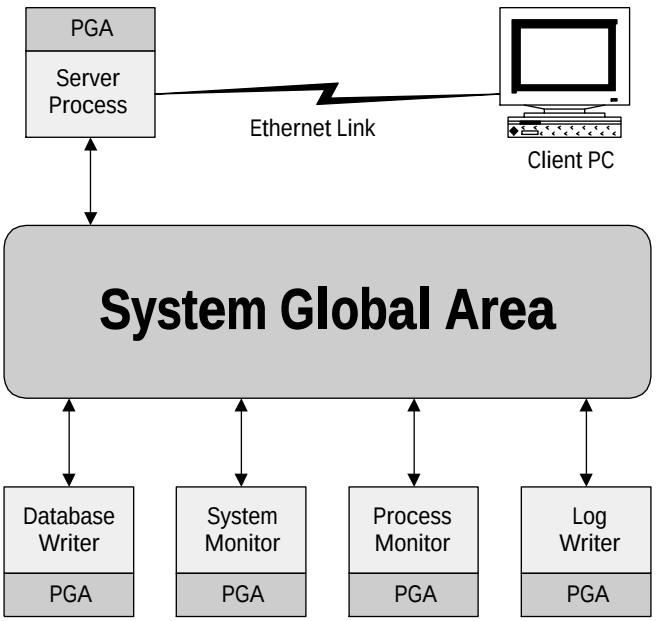


Figure 3-5: All processes share the SGA, and each process has its own PGA.

Understanding the System Global Area

The SGA is the most significant memory structure in an Oracle instance. It is composed of several major components: the database buffer cache, the shared pool, the redo log buffer, the large pool, and the fixed SGA. These are shown in Figure 3-6.

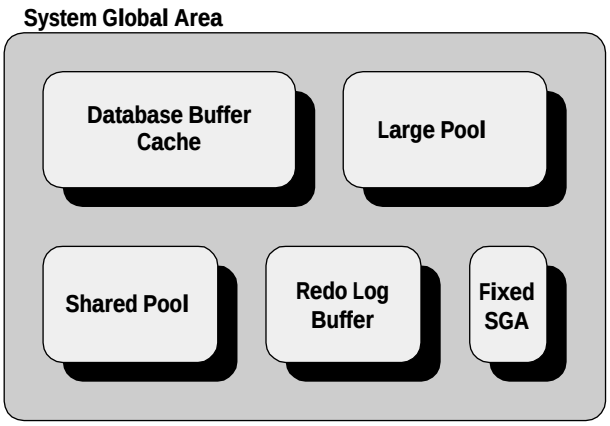


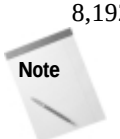
Figure 3-6: Oracle's system global area

Properly sizing the structures in the SGA is absolutely critical to proper database performance. You control their size. To properly size the structures in the SGA, you need to understand how they are used. The next few sections explain the use of each SGA structure. For suggestions on sizing these structures, see Chapter 20, "Database and Instance Tuning."

Understanding the database buffer cache

If Oracle had to read every block of data from disk each time you executed a query and had to write each block back to disk after you changed it, then Oracle would be a slow database indeed. Instead, Oracle caches frequently used data blocks in memory where they can be accessed quickly. The area in memory used to store frequently accessed data is known as the *database buffer cache*.

The database buffer cache consists of a number of buffers in memory. The size of each buffer matches the database block size. As blocks are read from disk, they are placed into the buffers. The number of buffers in the buffer cache is controlled by the `db_block_buffers` parameter. Thus, if your parameter file specifies `db_block_buffers = 8192`, then Oracle will reserve enough memory to hold 8,192 database blocks.



The size of a database block is controlled by the `db_block_size` parameter. To get the size of the buffer cache in bytes, multiply the `db_block_size` value by the `db_block_buffers` value. If your block size is 4,096, and you have 8,192 buffers in the cache, then the total size of the cache is 33,554,432 bytes.

Buffer Pools

The database buffer cache is frequently the largest part of the SGA. It consists of three smaller structures known as *buffer pools*, each of which is used to cache data with different access characteristics. Figure 3-7 shows these three buffer pools.

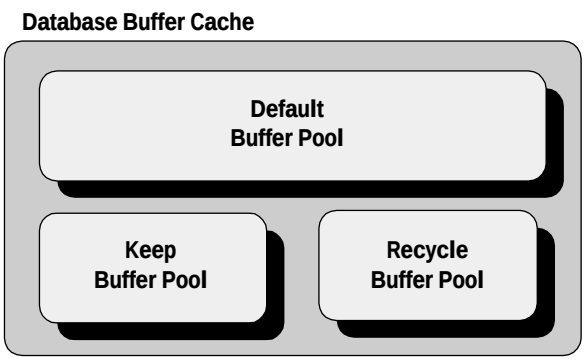


Figure 3-7: The buffer pools in the database buffer cache

Two of the buffer pools, the *keep buffer pool* and the *recycle buffer pool,* are optional. Every instance has at least one buffer pool — the *default buffer pool.* The buffer pools serve the following purposes:

keep buffer pool	Use the keep buffer pool for frequently accessed schema objects, such as code tables, that you want to keep in memory all the time. Data read into the keep buffer pool is retained until you shut down the database. It is never aged out of memory to make room for new data.
recycle buffer pool	Use the recycle buffer pool for schema objects that you want flushed out of memory as quickly as possible. A large table that you frequently scan in its entirety would be a good candidate for the recycle buffer pool.
default buffer pool	Use the default buffer pool for all objects that don't fall into the keep or recycle category.

Prior to the release of Oracle8, Oracle supported only one buffer pool. It was the equivalent of the default buffer pool, but it wasn't named because there was no need to distinguish it from other buffer pools.

Sizing buffer pools

The `buffer_pool_keep` and the `buffer_pool_recycle` parameters control the size of the keep and recycle buffer pools. The size of the default buffer pool is what's left over after subtracting the size of the keep and recycle pools from the total size of the buffer cache. Let's take a look at a sample configuration:

```
db_block_buffers         8192
buffer_pool_keep         2000
buffer_pool_recycle        1000
                         ----
size of DEFAULT pool     5192
```

In this example, the total size of the database buffer cache is 8,192 buffers. You set this by using the `db_block_buffers` parameter. Of those 8,192 buffers, 2,000 were allocated to the keep buffer pool. Another 1,000 were allocated to the recycle buffer pool. This leaves 5,192 for the default buffer pool.

Assigning objects to a buffer pool

Database objects, such as tables and indexes, are assigned to a buffer pool when you create them. You must decide which buffer pool is most appropriate. You make the buffer pool assignment using the `STORAGE` clause of the `CREATE` statement, as shown in this example:

```
CREATE TABLE state_codes (
    state_abbr        varchar2(2),
    state_name        varchar2(30)
) TABLESPACE users
STORAGE (BUFFER_POOL KEEP);
```

Here, the `state_codes` table was assigned to the keep buffer pool. This makes sense because the `state_codes` table won't be very big, and in all likelihood, it will be a frequently referenced table.

Dirty Lists and the LRU Lists

Oracle uses two lists to manage each buffer pool: a dirty list and a least recently used (LRU) list. The *dirty list* keeps track of which buffers in the pool have been modified and are in need of being written back to disk. The LRU list keeps track of how recently each buffer has been accessed. Figure 3-8 shows how these lists might look in relation to the DEFAULT buffer pool.

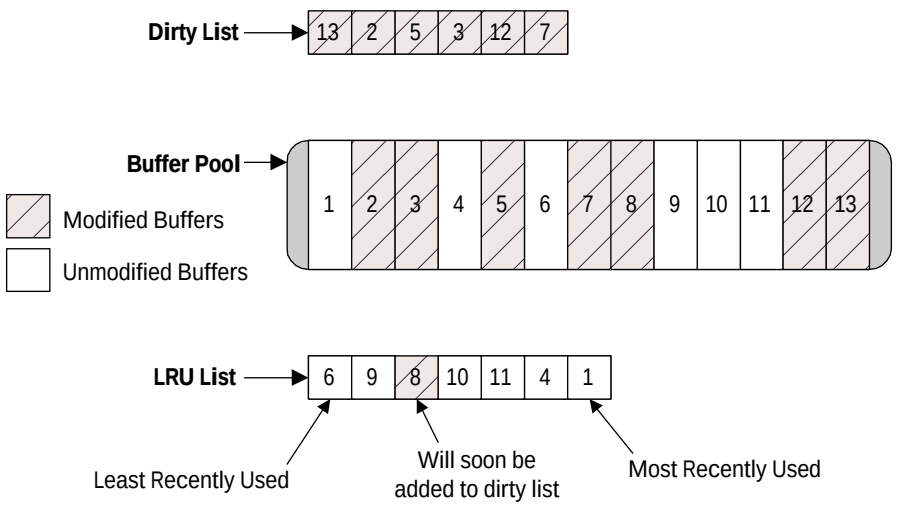


Figure 3-8: The dirty list and the LRU list keep track of buffers in a buffer pool.

Oracle uses the LRU list to decide which buffers to overwrite when new data needs to be read in from disk. The LRU list has two ends: a least recently used (LRU) end and a most recently used (MRU) end. Every time a buffer is accessed to satisfy a SQL statement, Oracle moves the pointer to that buffer to the most recently used end of the LRU list. This results in the LRU list always containing a list of buffers in the order in which they have recently been accessed.

When Oracle needs to read new data from disk, it starts at the least recently used end of the LRU list and looks for a buffer that hasn't been modified. When it finds one, the newly read data is placed in that buffer. Pointers to frequently accessed data blocks will tend to migrate to the most recently used end of the LRU list, and consequently, they will be the last to be overwritten. Keeping the most frequently used data in memory is a great asset in terms of performance. Memory is much faster than disk.

The dirty list is used to keep track of which buffers have been changed and need to be written back to disk. Whenever a buffer is modified, usually as a result of a SQL statement, Oracle will mark the buffer as *dirty*. Dirty buffers are quickly added to the dirty list. The database writer background processes, which you will read more about later in this chapter, check the dirty list regularly, and write those modified blocks back to disk.

Buffers aren't always added to the dirty list immediately when they are modified. Oracle tends to perform processes asynchronously, and it is possible for dirty buffers to remain in the LRU list for a short period of time. Ultimately, though, they'll be moved to the dirty list and then written to disk.

Caching in the shared pool

The *shared pool* is an area in memory where Oracle caches PL/SQL program units, parsed versions of SQL statements, execution plans for those parsed SQL statements, and data dictionary information. Like the database buffer cache, the shared pool is a major piece of the SGA, and its size has a significant impact on database performance.

Components of the Shared Pool

The two major components of the shared pool are the library cache and the data dictionary cache. The library cache is further subdivided into the shared SQL area and the PL/SQL area. Figure 3-9 shows these structures in the shared pool.

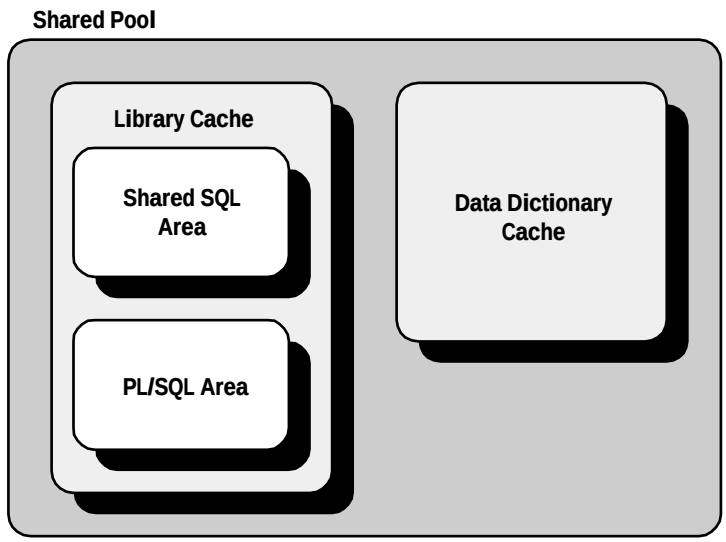


Figure 3-9: Memory structures in the shared pool

In addition to what is shown in Figure 3-9, the shared pool contains several relatively minor items such as locks, library cache handles, and memory-for-character-set conversion.

Sizing the Shared Pool

The `shared_pool_size` parameter controls the size of the shared pool. You can set the size at 50 million bytes, for example, by placing the following line in your database's parameter file:

```
shared_pool_size = 50000000
```

You can also use M and K prefixes to specify the size in megabytes or kilobytes.

Oracle determines the size of all the structures within the shared pool automatically based on the total size that you specify. You have no control over how much shared pool space gets allocated to the library cache vs. the data dictionary cache. Oracle determines that.

The Library Cache

The Library Cache contains the shared SQL area and the PL/SQL area, which work similarly. The shared SQL area holds parsed versions of SQL statements so that they don't have to be reparsed if they are used again. The PL/SQL area holds the compiled versions of PL/SQL procedures, functions, packages, and other program units so that all the database users can share them.

The shared SQL area

The shared SQL area holds parsed versions of SQL statements that database users have executed. The shared SQL area also holds the execution plans for those statements. The purpose is to speed the process along if and when those statements are reused.

Think for a minute about what Oracle has to do when you execute a SQL statement. First it has to parse the statement. *Parsing* refers to the process of taking the syntax of the statement apart, verifying that it is correct, and validating that the table and column names used in the statement are correct. Parsing can be an expensive operation in terms of time and disk I/O because Oracle needs to read information from the data dictionary to parse a SQL statement. To get that information, Oracle actually issues SQL statements internally. These are referred to as *recursive SQL* statements, and they too must be parsed.

After a statement has been parsed and Oracle understands what you want to do, Oracle must then figure out *how* to do it. It must build an *execution plan* for the statement. Figure 3-10 provides an example of one. Given the `SELECT` statement on the left, Oracle might come up with the execution plan shown on the right. Building the execution plan might involve even more recursive SQL statements, and Oracle

often needs to consider several possible plans before it can determine which is most efficient.

Query

```
SELECT id_no, animal_name
FROM aquatic_animal a
WHERE NOT EXISTS {
    SELECT *
    FROM checkup_history ch
    WHERE ch.id_no = a.id_no
    AND ch.checkup_date >
add_months(trunc(sysdate),-12));
```

Execution Plan

```
0 SELECT STATEMENT   Cost = 1
1   FILTER
2     TABLE ACCESS FULL AQUATIC_ANIMAL
3     TABLE ACCESS FULL CHECKUP_HISTORY
```

Figure 3-10: A SQL statement and its execution plan

Although all this parsing and execution-plan building is expensive, you can short-circuit much of it. Typically, the programmers define the SQL statements used by any given application when they write that application. You may have a lot of people using an application, but they will all be executing the same SQL statements over and over again. The people developing the Oracle software recognized that they could gain efficiency by simply saving the parsed SQL statements together with their execution plans. When the statement is next executed, Oracle simply needs to retrieve the preexisting plan. What a savings! No parsing. No rebuilding of the execution plan. No recursive SQL.

The shared SQL area is the part of the SGA that stores parsed SQL statements and their execution plans. Being able to reuse execution plans is critical to good database performance. Consequently, it's important to size the shared pool so that the shared SQL area is large enough to hold all the SQL statements that you use regularly. Chapter 20, "Database and Instance Tuning," provides some techniques that you can use to determine whether your shared SQL area is large enough.

The PL/SQL area

The PL/SQL area serves much the same purpose for PL/SQL code as the shared SQL area does for SQL statements. It allows multiple users to share the compiled version of one PL/SQL program unit.

When you execute a PL/SQL program unit, such as a trigger or a stored procedure, Oracle must load the compiled version of that program unit into memory. Sometimes, especially with PL/SQL packages, a program unit can be quite large. If a second user comes along and needs to execute the same trigger, stored procedure, or function, you don't want to have to load that same code into memory twice. Doing so costs both disk I/O and memory usage. To avoid this, Oracle loads compiled PL/SQL code into an area of the Library Cache set aside for that purpose. If two people execute the same code, they will both share the same copy.

While users share copies of PL/SQL code, they don't share copies of the variables. Each user actually gets his or her own private PL/SQL area where any variables are stored. This is what enables many users to execute the same code without conflicting with one another.

The Dictionary Cache

The dictionary cache is an area in the shared pool that Oracle uses to cache data dictionary information. Oracle frequently refers to the data dictionary when parsing SQL statements to verify table names, column names, datatypes, and so forth. By caching the most frequently used data dictionary information in memory, Oracle reduces the performance hit caused by recursive SQL statements.

The dictionary cache is sometimes referred to as the row cache.

Writing to the redo log buffer

The *redo log buffer* is an area in memory where Oracle places redo log entries that need to be written to disk. Every Oracle database has a background process called the log writer that constantly checks for new redo log entries and writes those entries to disk as quickly as possible. However, disk I/O is a lot slower than memory, and the log writer process can't keep up during heavy bursts of activity. The buffer evens things out. When changes are coming thick and fast, redo log entries are added to the buffer faster than they can be written, and the buffer starts to fill up. When the rate of redo generation subsides, the log writer process will catch up, and the buffer will empty out. For good database performance, the redo log buffer needs to be large enough to accommodate any sudden burst of activity, and the overall rate of redo log generation needs to be in a range that the log writer can handle.

Organization of the Redo Log Buffer

The redo log buffer is a first-in, first-out buffer that Oracle uses in a circular fashion. Oracle maintains two pointers, one pointing to the head of the log and the other pointing to the tail of the log. See Figure 3-11.

As changes are made to the database, Oracle always adds redo entries onto the head of the log, advancing the head pointer each time. The log writer process, which writes these entries to disk, always writes from the tail of the log. Thus, the tail chases the head in a circular fashion. In Figure 3-11, the buffer contains 11 entries. As each pointer advances past entry 11, it will be reset to point at entry 1, and the process of advancing through the buffer will begin all over again. This continues for as long as the database is running.

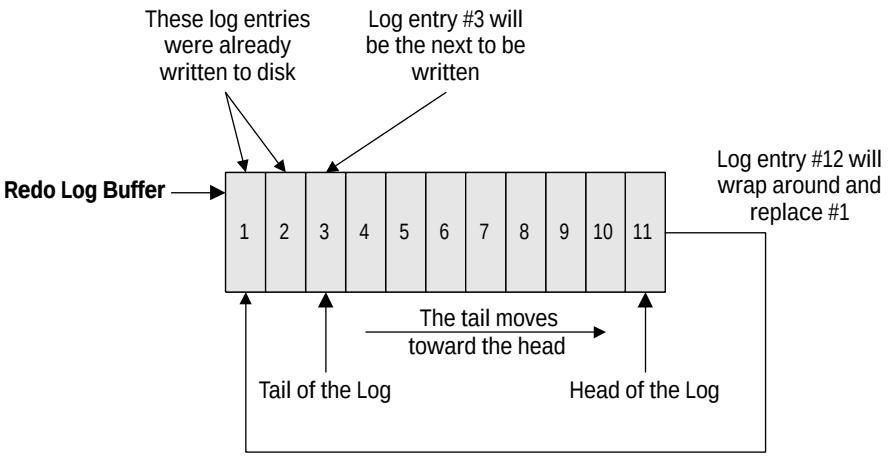


Figure 3-11: The redo log buffer

Sizing the Redo Log Buffer

You use the `log_buffer` parameter to size the redo log buffer. To allocate a 1MB redo log buffer, you would place the following entry in your database's parameter file:

```
log_buffer = 1024000
```

You want the redo log buffer to be large enough to accommodate normal bursts of activity that occur during daily operations. Chapter 20, "Database and Instance Tuning," tells you how to monitor for problems related to the redo log buffer size.

Using the large pool

The large pool is an optional feature introduced with the release of Oracle8 that provides a separate memory area where large blocks of memory can be allocated. You don't have to have one, but it's a good idea if you use either the multithreaded server option or Oracle's `RMAN` utility for backup and restore operations.

Large Pool Uses

Oracle's multithreaded server option uses the large pool as a place to allocate session memory. Session memory tends to be quite large when a multithreaded server is being used, and it works better to allocate that memory in the large pool. The large pool is also used by Oracle's `RMAN` utility to allocate I/O buffers used for backup and restore operations.

If you don't allocate a large pool, memory for user sessions and backup and restore operations end up being allocated from the shared pool. Oracle may sometimes be forced to reduce the amount of memory available for caching SQL statements to allocate enough memory for a multithreaded server session or for a backup operation. This can have a negative impact on performance.

Sizing the Large Pool

You size the large pool using the `large_pool_size` initialization parameter. The default size is zero. The minimum size is 600KB. The maximum size is operating-system specific but will always be at least 2GB. To allocate a 600KB large pool, you would place the following line in your database's parameter file:

```
large_pool_size = 600K
```

If you don't explicitly set the large pool's size using the `large_pool_size` parameter, Oracle will default to not using a large pool at all.

Understanding the fixed SGA

The fixed SGA is an area in the SGA that Oracle uses to store the myriad number of values that it needs to keep track of internally for the instance to operate. You can't size the fixed SGA. You don't need to tune the fixed SGA; just be aware that it exists.

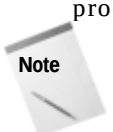
Understanding program global areas

In addition to the shared memory available in the SGA, each process connected to an Oracle database needs a private memory area of its own. Oracle refers to this area as a program global area (PGA). Processes use PGAs to store variables, arrays, and other information that do not need to be shared with other processes.

Contents of the PGA

The exact contents of the PGA depend on whether you are using Oracle's multithreaded server option. Figure 3-12 shows the contents of the PGA in both the standard and the multithreaded server configurations.

The session information box is sometimes referred to as the user global area (UGA) because it contains information specific to a particular database user connection. When a multithreaded server is being used, the UGA is stored in either the shared pool or the large pool because it must be accessible to more than one server process.



Under the multithreaded server configuration, the server process handling a user's SQL statements may change from one statement to the next: hence, the need for the UGA to be accessible to more than one process.

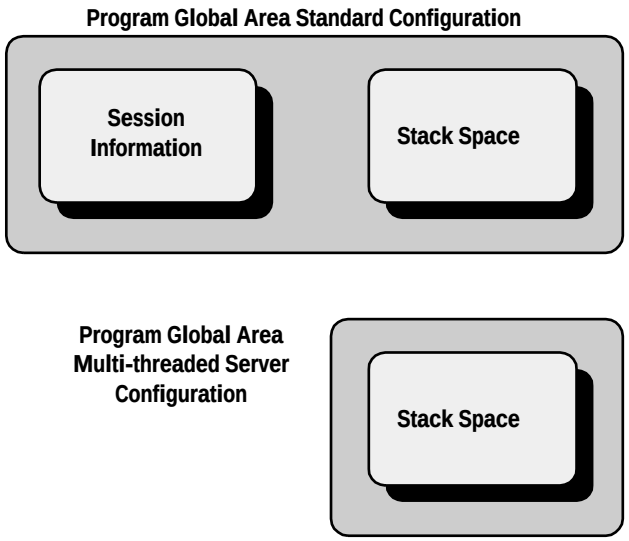


Figure 3-12: Contents of the PGA

Sizing the PGA

The `sort_area_size` and `sort_area_retained_size` parameters have the greatest effect on the size of the PGA. Together, these parameters control the amount of memory available to a process for sorting. Under the standard configuration, the entire sort area is contained in the PGA, so the PGA's size can potentially range up to the size specified by the `sort_area_size` initialization parameter. Actually, it can exceed that, because there are other structures in the PGA as well.

The maximum sort area space is not allocated unless required. If the sort area size is 10MB, and the largest sort you do requires only 2MB, then only 2MB will be allocated.

When you are using the multithreaded server option, the retained portion of the sort area is allocated in the SGA, and the amount allocated in the PGA will be equivalent to `sort_area_size` – `sort_area_retained_size`.

The sort area isn't the only structure in the PGA, and the sort area parameters aren't the only ones that affect the size of the PGA. The `open_links` and `db_files` parameters also affect the size of the PGA. However, as far as tuning goes, you need to worry about the sort area parameters.

Looking at Process Architecture

An Oracle instance is composed of processes and memory structures. You've already learned about the memory part; now it's time for the processes. An Oracle instance is composed of a number of processes called *background processes.* They're called background processes because they are always running, whether or not any users are connected to the database. Figure 3-13 shows a typical collection of background processes for an instance.

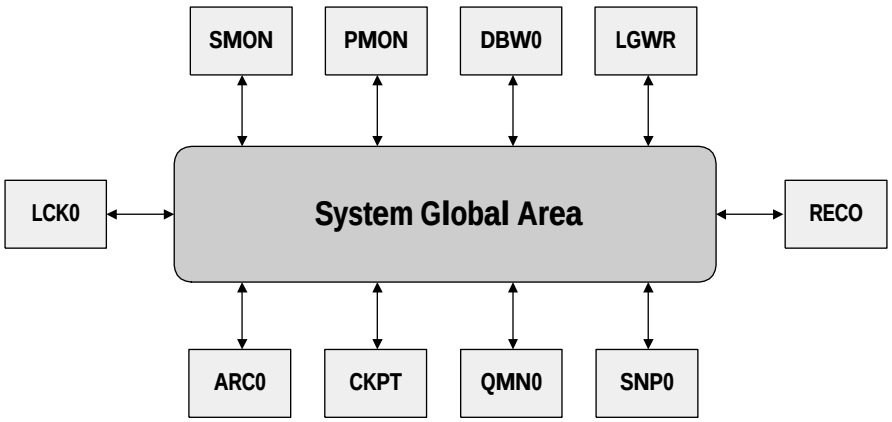


Figure 3-13: A typical Oracle instance

Each of the processes shown in Figure 3-13 has a specific job to do. You'll read more about each process later. For now, here's a brief synopsis of each process's function:

Database Writer (DBW0)	Database Writer processes write-modified data blocks back to the datafiles.
Log Writer (LGWR)	Log Writer processes write-redo log entries to the redo log files.
System Monitor (SMON)	System Monitor processes perform crash recovery and coalesces free space.
Process Monitor (PMON)	Process Monitor processes watch for processes that are prematurely disconnected, release any locks that they hold, and take care of any other necessary cleanup tasks.
Recoverer (RECO)	Recoverer processes resolve distributed transactions.

Snapshot (SNP0)	Snapshot processes run jobs from the database job queue.
Queue Monitor (QMN0)	Queue Monitor processes are used by the Advanced Queueing option to manage message queues.
Lock (LCK0)	Lock processes are used by the Parallel Server option to manage interinstance locking.
Checkpoint (CKPT)	Checkpoint processes periodically checkpoint the database. Checkpointing is the process of recording the current system change number in all of the database files.
Archiver (ARC0)	Archiver processes copy filled redo log files to the archive log destination.

Not all processes will be present for every Oracle instance. Some of them are optional, such as the SNP0 processes. You will see them only if you have set the `job_queue_processes` initialization parameter to a value greater than zero.

Using the appropriate operating system commands, you can list the processes that are running for any given Oracle instance. This is sometimes useful as a quick check to be sure that the instance is up and running.

Oracle processes under UNIX

On most UNIX systems, you can use the `ps` command to list processes. To see the Oracle background processes, you'll want to use `ps -ef` to get an extended process listing. Unless you want to wade through a list of all the processes running on your computer, you'll also want to use `grep` to search for the instance name. The following example shows how you would generate a list of all the background processes running for the `PROD` instance:

```
$ ps -ef | grep PROD
  oracle  3592     1  0 17:00:05 ?          0:00 ora_smon_PROD
  oracle  3590     1  0 17:00:05 ?          0:03 ora_lgwr_PROD
  oracle  3588     1  0 17:00:05 ?          0:05 ora_dbw0_PROD
  oracle  3586     1  0 17:00:05 ?          0:03 ora_pmon_PROD
  oracle  3594     1  0 17:00:05 ?          0:00 ora_reco_PROD
  oracle  3597     1  0 17:00:05 ?          0:03 ora_arc0_PROD
```

You can use a variation of this technique to quickly see which databases are running. For instance, instead of using `grep` to search for an instance name, use

grep to search for one of the mandatory processes instead. You'll get a line of output for each instance that is currently running. Here's an example:

```
$ ps -ef | grep smon
  oracle  2564     1  2 07:00:01 ?          0:05 ora_smon_TEST
  oracle  2588     1  2 07:00:05 ?          0:05 ora_smon_DEVL
  oracle  2576     1  1 07:00:03 ?          0:06 ora_smon_PROD
  oracle  3351  3336  1 17:19:02 ttyp2      0:00 grep dbw0
```

This example searched for the SMON process. You could just as easily use grep for LGWR or DBW0. Be careful with the numbered processes, though. If you use grep for DBW9 and not all instances are configured to run nine database writers, you'll miss some.

Oracle processes under Windows NT

Under Windows NT, the Oracle processes are implemented as threads that run within a service. Oracle provides a utility, the Database Administration Assistant for Windows NT, that allows you to view a list of these threads. You can find this utility on the Database Administration menu, which you get to by selecting Start and pointing to Programs and Oracle – OraHome1. When you run it, you can navigate to the instance you want, right-click it, choose Process Information from the pop-up menu, and you'll see a screen like the one shown in Figure 3-14.

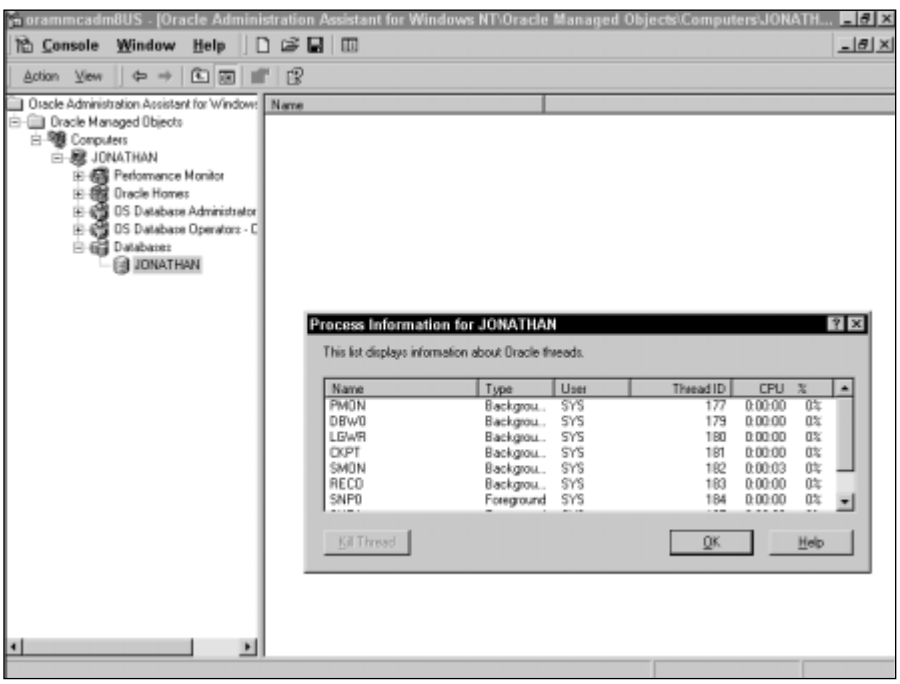


Figure 3-14: Oracle Administration Assistant for Windows NT showing the threads in an Oracle instance

In order to use the Oracle Administration Assistant for Windows NT, you need to install the Microsoft Management Console version 1.1. You can download it from Microsoft's website.

If you need to know which instances are running, your best bet is to look at the Services control panel. It will tell you which Oracle database services are running, although it's still possible for an instance to be shut down even though its service is running.

Database Writer processes

An Oracle instance can have up to ten Database Writer processes (DBW0). These will be numbered `DBW0` through `DBW9`. The Database Writer's job is to write modified data blocks back to disk.

Every instance gets one Database Writer by default. You can change the number of Database Writer processes by using the `db_writer_processes` initialization parameter. To configure an instance for the full complement of ten Database Writer processes, add the following line to the parameter file:

```
db_writer_processes = 10
```

Multiple Database Writers make sense only in a system with multiple CPUs because they allow you to spread the task of writing data evenly over those CPUs. If you are running on a single-CPU system, you should use just one Database Writer process.

The Log Writer process

The Log Writer's job is to write redo log entries to the database's online redo log files. Recall that as changes are made to a database, Oracle places redo log entries into the redo log buffer. For the Log Writer process (LGWR), these entries are always added to the head of the log. The Log Writer pulls these entries off the tail of the log and writes them to disk.

Archiver processes

Archiver processes (ARC0) have the task of copying filled redo log files to the archive log destination. An Oracle database has a finite number of redo log files, and Oracle writes to these in a circular fashion. It fills up each online redo log file in sequence, and when it gets to the end, it circles around and starts filling up the first redo log file again. If you want to save the log files for possible use in recovering the database, you need to make a copy of each log file before it is reused. You can do this manually, or you can start one or more archiver processes to automate the process.

As with database writers, Oracle allows you to have up to ten Archiver processes. The `log_archive_max_processes` parameter is used to set the maximum number that you want to allow. For example, the following entry in your database parameter file will set a maximum of five Archiver processes:

```
log_archive_max_processes = 5
```

Unlike the case with database writers, Oracle won't necessarily start five archiver processes just because you tell it to. Instead, it automatically starts and stops archiver processes as necessary to keep up with the amount of redo being generated. As the name indicates, the `log_archive_max_processes` parameter sets an upper limit on the number of archiver processes that Oracle can start.

The Checkpoint process

The Checkpoint process (CKPT) is responsible for recording Checkpoint information in all database file headers. Periodically, when an instance is running, Oracle records a Checkpoint in all database file headers indicating the most recent redo log entry for which all changes have been written to the database files. Oracle uses this information during a recovery operation to determine which log files must be read and which log entries must be reapplied.

The System Monitor process

The System Monitor process (SMON) has three functions in life:

- ✦ Crash recovery
- ✦ Cleanup of temporary segments
- ✦ Coalescing free space

Crash recovery happens when you restart the database after a system crash or an instance crash. When a crash occurs, you will very likely lose changes that were buffered in memory, but not written to disk, before the crash occurred. Some of these changes might represent committed transactions. When the instance is restarted, SMON recovers these transactions by reapplying the lost changes based on the information in the online redo log files. This works because Oracle always ensures that all changes for a committed transaction have been recorded in the redo log and that those log entries have been physically written to disk.

Another of SMON's tasks, and a more mundane one, is to deallocate temporary segments used for sorting. When you issue a `SELECT` statement, or any other SQL statement that requires a lot of data to be sorted, the sort may not be done entirely in memory. When too much data exists to be sorted in memory, Oracle sorts a piece at a time and uses disk space to temporarily hold the results. When the sort is over, it is SMON's task to deallocate this space.

Coalescing free space is the last of SMON's major functions. For tablespaces with a default PCTINCREASE setting that is greater than 0, SMON continuously checks data files, looking for two or more adjacent areas of free space. Whenever it finds adjacent areas of free space, SMON combines them into one larger area. This helps avoid fragmentation and also lets you allocate larger extents that might not be possible otherwise.

The Process Monitor process

The Process Monitor (PMON) performs the same kind of tasks for processes that the System Monitor does for an instance. The difference is that instead of cleaning up after an instance crash, the PMON cleans up after processes that abnormally terminate. When a process aborts, PMON does the following:

- ✦ Releases any locks held by the process
- ✦ Rolls back any transactions that the process had started but had not yet committed
- ✦ Removes the process ID from the list of active processes

The Recoverer process

The Recoverer process (RECO) resolves distributed transactions that have failed. This process will be present only if the `distributed_transactions` initialization parameter is greater than 0, indicating that the database supports distributed transactions. If a distributed transaction fails, Recoverer will communicate with the other nodes involved with the transaction to either commit or roll back the transaction.

Job queue processes

Job queue processes (SNP0) are used to run scheduled PL/SQL jobs. If you are using Oracle's replication features and you have created snapshots that refresh automatically at predefined intervals, a job queue process makes that happen. Job queue processes also run jobs that have been scheduled using the built-in DBMS_JOBS package.

The `job_queue_processes` initialization parameter controls how many of these processes are started, and you can have up to 36. The following entry, placed in a parameter file, would cause the maximum of 36 job queue processes to be started:

```
job_queue_processes = 36
```

The 36 job queue processes are named `SNP0` through `SNP9`, and then `SNPA` through `SNPZ`.

Queue Monitor processes

Queue Monitor processes (QMNO) are used with Oracle's Advanced Queuing option. You can have up to ten Queue Monitor processes, and they are configured using the `aq_tm_processes` initialization parameter. The following parameter file entry would configure an instance to have three Queue Monitor processes:

```
aq_tm_processes = 3
```

Queue Monitor processes are named `QMNO` through `QMN9`, depending on how many you create.

Summary

In this chapter, you learned:

- An Oracle instance consists of a set of processes and an area of shared memory. A database consists of files that contain related data.
- Three types of files make up an Oracle database: datafiles, log files, and control files.
- The system global area (SGA) is a large memory structure that Oracle background processes use.
- The major components of the SGA are the database buffer cache, the shared pool, the redo log buffer, and the large pool.
- Several processes combine to make an Oracle instance. Each has a specific function to perform.

✦ ✦ ✦