**Importance of Architecture in Oracle**

**Understand Components of Database**

Understanding the architecture of a running Oracle database instance is one of the key differentiators between an outstanding DBA and a run-of-the-mill DBA. Learn the architecture well. I am always surprised at how many people do not do this. Knowledge and understanding of how Oracle works behind the scenes is critical to the tuning process. It can also be very helpful when it comes to planning for backup and recovery. If you learn the architecture well, you will be able to solve problems that other people cannot, and you will be well on your way to certification and a successful career as a DBA.

[**Oracle Database Architecture**](https://amzn.to/3cQlqfK)

Oracle is designed to be a very portable database, it is available on every platform of relevance, from Windows to UNIX/Linux to mainframes. However, the physical architecture of Oracle looks different on different operating systems. For example, on a UNIX/Linux operating system, you will see Oracle implemented as many different operating system processes, virtually a process per major function. On UNIX/Linux, 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 nonscalable). On the Windows platform, Oracle is implemented as a single process with multiple threads. In the past, on IBM mainframe systems, running OS/390 and z/OS, the Oracle operating system's 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 the 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 that you can get a good understanding of how Oracle works on all platforms.

In this module, I present a broad picture of this architecture. We will take a look at the Oracle server and define some terms such as

1. database,
2. pluggable database,
3. container database, and
4. instance (terms that always seem to cause confusion).

**How the Server manages memory?**

We will take a look at what happens when you connect to Oracle and, at a high level, how the server manages memory. In the subsequent three chapters, we will look in detail at the three major components of the Oracle architecture:

A *pluggable database* will be associated with a single container database at a time and is only indirectly associated with an instance; it will share the instance created to mount and open the container database. So, like a container database, a pluggable database can be associated with one or more instances at any point in time. Unlike a single-tenant database, however, an instance may be providing access to many (up to around 250) pluggable databases simultaneously.  
That is, a single instance may be providing services for many pluggable databases, but only one container or single-tenant database. 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 single database, which 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 container or single-tenant database) associated with it. Multiple pluggable databases, subordinate to the container database, can be open and accessible simultaneously but will all share the single instance created to open the container database.  
In most cases, the opposite is true as well: a container or single-tenant database will have only one instance working on it. However, in the special case of (RAC) Oracle Real Application Clusters, an Oracle option 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 us start by taking a look at a simple example. Let us say we have just installed Oracle 12c version 12.1.0.1 on our UNIX/Linux based computer. We did a software-only installation. No starter databases, with nothing just the software. The pwd command shows the current working directory, dbs (on Windows, this would be the database directory) and the

ls -l

command shows that the directory is empty. There is no init.ora file and no SPFILEs (stored parameter files;).

**Understanding Oracle Database Architecture**

**Database Creation**

An Oracle database is composed of files. There are several types of files, and it is important that you as the DBA understand the purpose of each file, as well as what it contains. This module will help you do that.

[**Oracle DB Architecture**](https://www.relationaldbdesign.com/database-architecture/module3/oracle-db-architecture.php)

When you have completed this module, you will be able to:

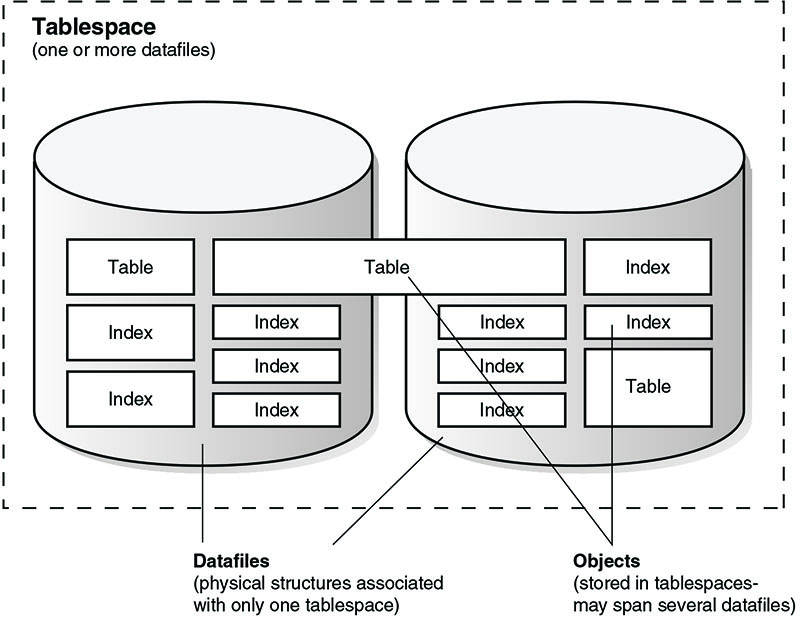
1. Identify the different files used by a database.
2. Edit and understand a database initialization file.
3. Explain the use and importance of redo logs for instance recovery.
4. Understand the relationship between tablespaces and datafiles.

**Tablespaces, Datafiles, and Control Files**

Oracle stores data logically in tablespaces and physically in datafiles associated with the corresponding tablespace. Figure 3-1 illustrates this relationship. Databases, tablespaces, and datafiles are closely related, but they have important differences:

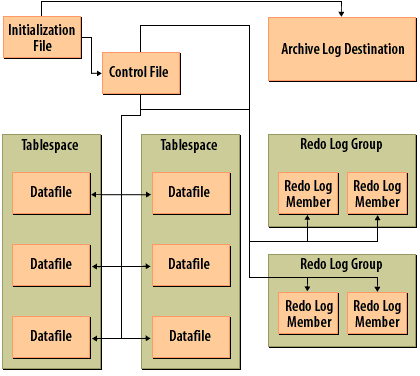
1. An Oracle database consists of one or more logical storage units called tablespaces, which collectively store all of the database's data.
2. Each tablespace in an Oracle database consists of one or more files called datafiles, which are physical structures that conform to the operating system in which Oracle is running.
3. The data of a database is collectively stored in the datafiles that constitute each tablespace of the database. For example, the simplest Oracle database would have one tablespace and one datafile. Another database can have three tablespaces, each consisting of two datafiles (for a total of six datafiles).

When ready, go to the next lesson, and begin.

Figure 3-1 describing Tablespace: 1) Datafiles: physical structures associated with only one tablespace, 2) Objects: stored in tablespaces, may span several datafiles

**Database File Overview**

An Oracle database involves files with several different purposes. Some files store data, some store information that Oracle needs to keep track of your data, and some store information needed to recover from a system crash. The following diagram shows the files that you will find in a typical Oracle database:



1. Initialization File: Contains parameters that specify the database block size, the amount of memory to use for the Shared Global Area (SGA), and that control other aspects of how the database instance operates.
2. Control File: Keeps track of all the files that make up a database.
3. Redo Log Member: Stores a log of all changes made to the database, the redo log, that is used in the event that the database needs to be recovered.
4. Tablespace: Tablespaces are logical storage structures that contain table and index data. Oracle allows you to map a tablespace onto one or more physical files.
5. Datafiles: These are the files that hold the data for tables and for indexes. They are the reason all the other files exist.
6. Redo Log Group: A group of one or more redo log files that Oracle treats as one. Oracle writes the same information to each file in a group.
7. Archive Log Destination: This is the location to which Oracle copies redo log files when they are filled.

Oracle Database Files: 1) Initialization file 2) Control File 3) Data File  
[**Database File Types**](https://www.relationaldbdesign.com/database-architecture/module3/db-file-types.php)

**Databases and Instances**

An Oracle database is a collection of data in one or more files. The database contains physical and logical structures. In the course of developing an application, you create structures such as tables and indexes to store rows and speed their retrieval. You can create synonyms for the object names, view objects in different databases (across database links), and restrict access to the objects. You can even use external tables to access files outside the database as if the rows in the files were rows in tables. In this book, you will see how to create these objects and develop applications based on them. An Oracle instance comprises a memory area called the System Global Area (SGA) and the background processes that interact between the SGA and the database files on disk. In an Oracle Real Application Cluster (RAC), more than one instance will use the same database. The instances generally are on separate servers connected by a high-speed interconnect.

**Inside the Database**

Within the Oracle database, the basic structure is a table. Oracle Database 11g supports many types of tables, including the following:

1. **Relational tables** Using the Oracle-supplied datatypes, you can create tables to store the rows inserted and manipulated by your applications. Tables have column definitions, and you can add or drop columns as the application requirements change. Tables are created via the create table command.
2. **Object-relational tables** To take advantage of features such as type inheritance, you can use Oracle's *object-relational* capabilities. You can define your own datatypes and then use them as the basis for column definitions, object tables, nested tables, varying arrays,
3. **Index-organized tables** : You can create a table that stores its data within an index structure, allowing the data to be sorted within the table.
4. **External tables:** Data stored in flat files may be treated as a table that users can query directly and join to other tables in queries. You can use external tables to access large volumes of data without ever loading them into your database. Note that Oracle also supports BFILE datatypes, a pointer to an external binary file. Before creating a BFILE or an external table, you must create a directory alias within Oracle (via the create directory command) pointing to the physical location of the file.
5. **Partitioned tables** :You can divide a table into multiple partitions, which allows you to separately manage each part of the table. You can add new partitions to a table, split existing partitions, and administer a partition apart from the other partitions of the table. Partitioning may simplify or improve the performance of maintenance activities and user queries. You can partition tables on ranges of values, on lists of values, on hashes of column values, or on combinations of those options.
6. **Materialized views** A materialized view is a replica of data retrieved by a query. User queries may be redirected to the *materialized views* to avoid large tables during execution.  
   The optimizer will rewrite the queries automatically. You can establish and manage refresh schedules to keep the data in the materialized views fresh enough for the business needs.
7. **Temporary tables**: You can use the *create global temporary table* command to create a table in which multiple users can insert rows. Each user sees only his or her rows in the table.
8. **Clustered tables:** If two tables are commonly queried together, you can physically store them together via a structure called a cluster.
9. **Dropped tables:** You can quickly recover dropped tables via the flashback table to before drop command. You can flash back multiple tables at once or flash back the entire database to a prior point in time. Oracle supports flashback queries, which return earlier versions of rows from an existing table.

**Oracle Initialization File**

**Parameters in the Initialization File**

The database initialization file is a text file containing a number of parameters that control how the Oracle database works. At a very high-level, you could categorize those parameters as:

1. A pointer to the database control files
2. Information about the database itself, such as the physical block size used when reading and writing datafiles
3. Parameters that control the Oracle *background processes*
4. Parameters that control how Oracle allocates memory
5. Pointers to the directories containing the alert log and trace files for the instance

Here are some lines from a typical *initialization file*:

db\_name = devl

db\_files = 1024

control\_files =

(/m01oracle/oradata/SCTPROD/control01.ctl,

/m21/oradata/SCTPROD/control02.ctl,

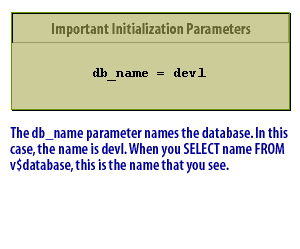
/m57/oradata/SCTPROD/control03.ctl)

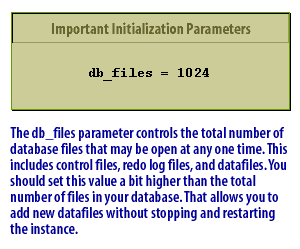
db\_block\_size = 8192

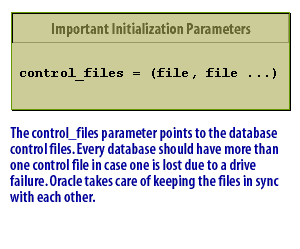
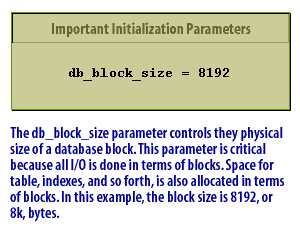
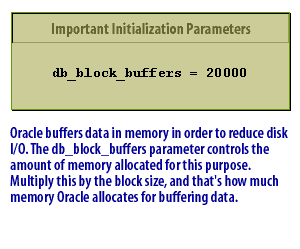
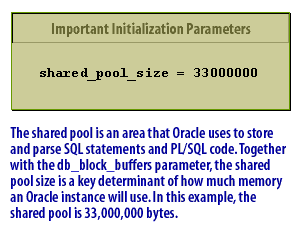
**Parameter Details**

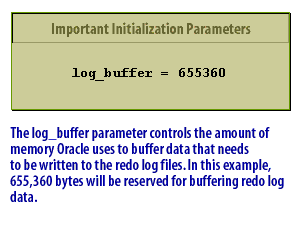
There are many parameters that you can set in the initialization file. Oracle documents all of them in the *Oracle Server Reference Manual*.  
Look through the following slideshow to learn about the most important parameters in a bit more detail:

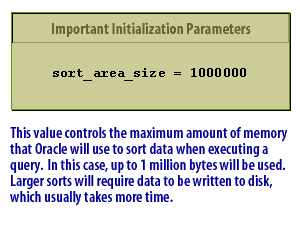
## Important Oracle Initialization Parameters

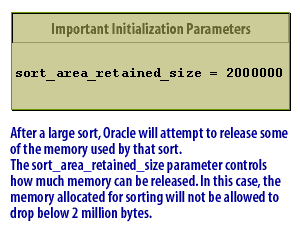


The db\_name parameter names the database. In this case, the name is dev1. When you SELECT name FROM v$database, this is the name that you see.  


The db\_files parameter controls the total number of database files that may be open at any one time. This includes control files, redo log files, and datafiles. You should set this value a bit higher than the total number of files in your database. That allows you to add new datafiles without stopping and restarting the instance.  
The control\_files parameter points to the database control files.  
The db\_block\_size parameter controls the physical size of a database block.  
Oracle buffers data in memory in order to reduce disk I/O.  


The shared pool is an area that Oracle uses to store and parse SQL statements and PL/SQL code.  


The log\_buffer parameter controls the amount of memory Oracle uses to buffer data that needs to be written to the redo log files.  


This value controls the maximum amount of memory that Oracle will use to sort data when executing a query.  


After a large sort, Oracle will attempt to release some of the memory used by that sort.

### How the initialization file works

Oracle reads the initialization file whenever you start an instance.  
Any changes that you make to the initialization file afterwards will not take effect until you shut down and restart your system.  
Strictly speaking, Oracle does not consider the initialization file to be part of a database.

**Oracle initialization file not a database file**

The initialization file is also often called a *parameter file*. It is a text file containing a number of parameter settings that affect how the Oracle processes operate when you start an instance and open a *database*. In that sense, the initialization file is more related to an Oracle instance than to a database. However, one of the most important parameters in the initialization file points to the database control files.  
For the most part, opening your database is not dependent on having one specific set of parameters. It is not unheard of for a site to have two initialization files, both used with the same database but for different purposes. One may be optimized for use during the day when most of the workload comes from small, interactive transactions. The other may be optimized for use at night, when the work comes from large, batch processes.

**initialization file**

Since the initialization file is read when an Oracle instance is started, changes to these parameters do not take effect until an instance is stopped and restarted. Remember, though, that turning on automatic archiving does not put the database in ARCHIVELOG mode. Similarly, placing the database in ARCHIVELOG mode does not enable the automatic archiving process. You should also make sure that the archive log destination has enough room for the logs Oracle will automatically write to it. If the archive logfile destination is full, Oracle will hang since it cannot archive additional redo logfiles.  
The *archived redo logs* are critical for *database recovery*. Just as you can duplex the online redo logs, you can also specify multiple archive log destinations. Oracle will copy filled redo logs to specified destinations and you can also specify whether all copies must succeed or not.  
The initialization parameters for this functionality are as follows:

LOG\_ARCHIVE\_DUPLEX\_DEST

Specifies an additional location for redundant redo logs.

LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST

Indicates whether the redo log must be successfully written to one or all of the locations. Valid values are 1 through 10 if multiplexing and 1 or 2 if duplexing. See your Oracle documentation for the additional parameters and views that enable and control this functionality.

**Initialization Parameters**

Initialization parameters are configuration parameters that affect the basic operation of an instance. The instance reads initialization parameters from a file at startup. Oracle Database provides many initialization parameters to optimize its operation in diverse environments. Only a few of these parameters must be explicitly set because the default values are usually adequate.

**Functional Groups of Initialization Parameters**

Most initialization parameters belong to one of the following functional groups:

1. Parameters that name entities such as files or directories
2. Parameters that set limits for a process, database resource, or the database itself
3. Parameters that affect capacity, such as the size of the SGA (these parameters are called variable parameters)

Variable parameters are of particular interest to database administrators because they can use these parameters to improve database performance.

**Basic and Advanced Initialization Parameters**

Initialization parameters are divided into two groups:

1. basic and
2. advanced.

Typically, you must set and tune only the approximately 30 basic parameters to obtain reasonable performance. The basic parameters set characteristics such as the database name, locations of the control files, database block size, and undo tablespace. In rare situations, modification to the advanced parameters may be required for optimal performance. The advanced parameters enable expert DBAs to adapt the behavior of the Oracle Database to meet unique requirements. Oracle Database provides values in the starter initialization parameter file provided with your database software, or as created for you by the Database Configuration Assistant.

You can edit these Oracle-supplied initialization parameters and add others, depending on your configuration and how you plan to tune the database. For relevant initialization parameters not included in the parameter file, Oracle Database supplies defaults.

It is an important file though, and you want to preserve it with the same care that you do all the other files that are database files.

**Oracle Database Control File**

**Information in the control file (Identify purpose and Contents)**

Every Oracle database contains a *control file*[[1]](https://www.relationaldbdesign.com/database-architecture/module3/database-control-file.php#fn1).

The control file gives Oracle a place to store important information about the physical state of the database. It contains the following information:

1. The database name
2. Information about tablespaces
3. The names and locations of all the datafiles
4. The names and locations of all the redo log files
5. The current log sequence number
6. Checkpoint information
7. Information about redo logs and the current state of archiving

**Main function of the control file**

Perhaps the most significant use of the control file is to keep track of all the other files that make up the database. When you start an instance and open a database, Oracle reads the initialization file to find the name and location of the control file. Then it reads the control file to find the names and locations of all the datafiles and redo log files. Finally, it opens all those files, making the database available for you to use.

**Importance of the control file**

The control file is so critical to the proper operation of an Oracle database that Oracle encourages you to keep at least three copies of the control file at all times. These should be on separate disks, and preferably separate *controllers* .

That is why the initialization file that you saw in the last lesson listed three control file names. The Oracle database software always writes the same information to all control files, thus keeping them in sync with one another.

[**Data Control Files**](https://www.relationaldbdesign.com/database-architecture/module3/data-control-files.php)

**Control Files**

Every Oracle database has a control file. For example, it contains the following information:

1. Database name
2. Names and locations of datafiles and redo log files
3. Time stamp of database creation

Oracle can multiplex the control file, that is, simultaneously maintain a number of identical control file copies, to protect against a failure involving the control file. Every time an instance of an Oracle database is started, its control file identifies the database and redo log files that must be opened for database operation to proceed. If the physical makeup of the database is altered (for example, if a new datafile or redo log file is created), then the control file is automatically modified by Oracle to reflect the change. A control file is also used in database recovery.

Control Files: The control files include information about the file structure of the database and the current log sequence number being written by LGWR. During normal recovery procedures, the information in a control file guides the automatic progression of the recovery operation.

[[1]](https://www.relationaldbdesign.com/database-architecture/module3/database-control-file.php#r1) *control file:*A control file contains entries that specify the physical structure of the database.

**Oracle Datafiles**

Datafiles typically comprise the largest part of any Oracle database, at least in terms of the disk space used. Datafiles are where Oracle stores your data. You can get a quick list of all the datafiles in your database and their sizes, by querying the V$DATAFILE view. The following example shows how to do this from Server Manager or from SQL\*Plus:

SQL> COLUMN name FORMAT A35

SQL> COLUMN status FORMAT A6

SQL> COLUMN bytes FORMAT 999,999,999

SQL> SELECT name, status, bytes

2 FROM v$datafile;

NAME STATUS BYTES

-------------------------------- ------ ------------

C:\ORANT\DATABASE\SYS1ORCL.ORA SYSTEM 31,457,280

C:\ORANT\DATABASE\USR1ORCL.ORA ONLINE 3,145,728

C:\ORANT\DATABASE\RBS1ORCL.ORA ONLINE 5,242,880

C:\ORANT\DATABASE\TMP1ORCL.ORA ONLINE 2,097,152

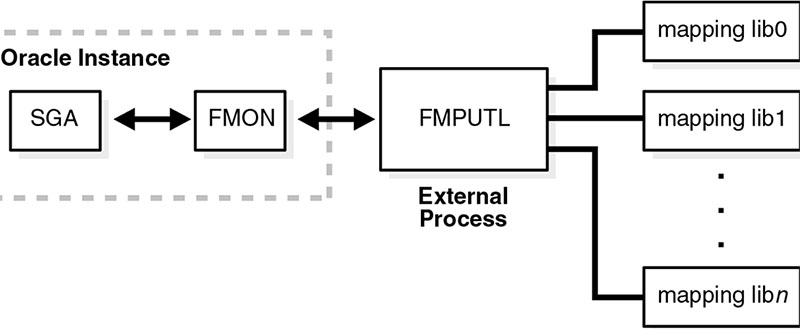
**Display of the datafile view**

If you do not mind a display with long lines that wrap around the screen, you can omit the COLUMN commands and just issue the SELECT statement. The STATUS column tells you whether the datafile is currently open for use, and also indicates whether a datafile belongs to the *system tablespace*.

A status of *ONLINE* tells you that the file is open. A status of *SYSTEM* indicates that the file belongs to the system tablespace.  
The system tablespace contains the *data dictionary* and is always online as long as an Oracle database is open. The BYTES column tells you how large the datafile is. In this example, the SYS1ORCL.ORA file is approximately 31 megabytes in size.

**How the Oracle Database File Mapping Interface Works**

This section describes the components of the Oracle Database file mapping interface and how the interface works. It contains the following topics:

Figure 2-5: Components of File Mapping  
The following sections briefly describes these components and how they work together to populate the mapping views:  
FMON is a background process started by the database whenever the FILE\_MAPPING initialization parameter is set to TRUE. FMON is responsible for:

1. Building mapping information, which is stored in the SGA. This information is composed of the following structures:
   1. Files
   2. File system extents
   3. Elements
   4. Sub elements
2. Refreshing mapping information when a change occurs because of:

a) Changes to data files (size)

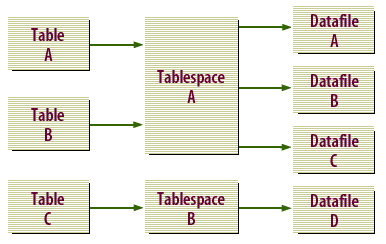
b) Addition or deletion of data files

c) Changes to the storage configuration (not frequent)

1. Saving mapping information in the data dictionary to maintain a view of the information that is persistent across startup and shutdown operations
2. Restoring mapping information into the SGA at instance startup. This avoids the need for a potentially expensive complete rebuild of the mapping information on every instance startup. You help control this mapping using procedures that are invoked with the DBMS\_STORAGE\_MAP package.

**What is tablespace?**

While Oracle physically stores 1) table and 2) index data in datafiles, it manages the relationship between tables, indexes, and datafiles using a logical concept called the tablespace. A tablespace is a grouping of one or more datafiles. When you create a table or an index, you need to tell Oracle where to store that object. You do that by providing a tablespace name. In turn, the tablespace definition includes a list of one or more datafiles, which is where Oracle actually stores the data. Move your mouse over the following diagram to see how this works:

Mapping of Tables to Table Data to Tablespaces

1. Table A: Table A's data is stored in Tablespace A
2. Table B: Table B's data is stored in Tablespace A.
3. Table C: Table C's data is stored in Tablespace B.
4. Tablespace A: Tablespaces may contain data from several objects, in this case Tables A and B.
5. Tablespace B: Tablespace B contains only data from Table C.
6. Datafile A: May contain data from Table A or Table B or both.
7. Datafile B: May contain data from Table A or Table B or both.
8. Datafile C: May contain data from Table A or Table B or both.
9. Datafile D: Will only contain data from Table C.

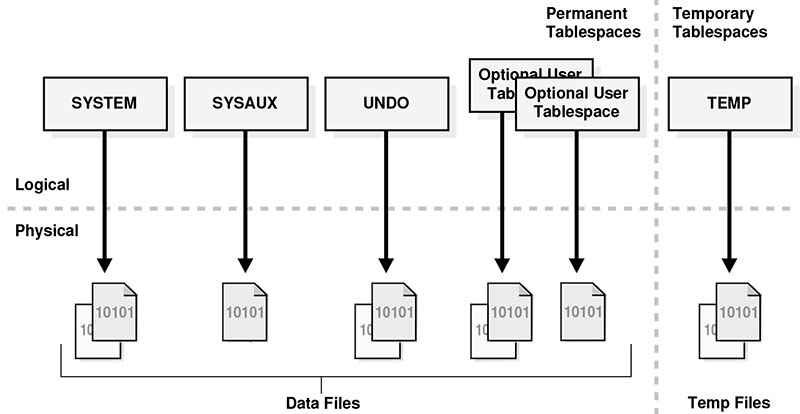
[**Tablespaces Datafiles**](https://www.relationaldbdesign.com/database-architecture/module3/tablespaces-datafiles.php)  
**Note:** While a tablespace may contain many datafiles, a datafile can only be part of one tablespace.

**How tablespaces create flexibility**

You can see that the tablespace stands between the object (table, index, etc) that you have created, and the physical location where that object is stored. The benefit you get from this relationship is the flexibility to add datafiles as the objects in a tablespace grow. Tablespaces allow a many-to-many relationship between database objects and datafiles. Many tables and indexes may be stored in a datafile, and many datafiles may be used for one table or index. The advantage to this twofold: you can easily add new datafiles whenever you need more space, and you can spread your datafiles over multiple disks as a way to distribute the I/O load.

**Overview of Tablespaces**

A tablespace is a logical storage container for segments. Segments are database objects, such as tables and indexes, that consume storage space. At the physical level, a tablespace stores data in one or more data files or temp files. A database must have the SYSTEM and SYSAUX tablespaces. The following figure shows the tablespaces in a typical database. The following sections describe the tablespace types.

Figure 3-6: Tablespaces

**Extract Oracle Tablespace Information**

**Viewing tablespace records**

There are two DBA views that return useful information about the tablespaces in a database.

1. DBA\_TABLESPACES
2. DBA\_DATA\_FILES

The first is DBA\_TABLESPACES. This view returns one record for each tablespace. That record contains the tablespace name, default storage parameters for objects stored in that tablespace, and a field indicating the current status of the tablespace. Use the following query to find out what tablespaces exist in your database:

SELECT tablespace\_name, status

FROM DBA\_TABLESPACES;

The results that you will get when you execute the above query will resemble these:

TABLESPACE\_NAME STATUS

------------------------------ ---------

SYSTEM ONLINE

USER\_DATA ONLINE

ROLLBACK\_DATA ONLINE

TEMPORARY\_DATA ONLINE

**Viewing datafiles with Tablespaces**

Another useful view is the DBA\_DATA\_FILES view. This view is similar to the V$DATAFILE view that lists the datafiles and their sizes, but DBA\_DATA\_FILES also returns the tablespace that each datafile is associated with. You can use the following query to get a list of datafiles used for each tablespace:

SELECT tablespace\_name, file\_name

FROM DBA\_DATA\_FILES

ORDER BY tablespace\_name, file\_name;

The results that you get when you execute the above query will resemble these:

TABLESPACE\_NAME FILE\_NAME

---------------- ---------------------------------

ROLLBACK\_DATA C:\ORAWIN95\DATABASE\RBS1ORCL.ORA

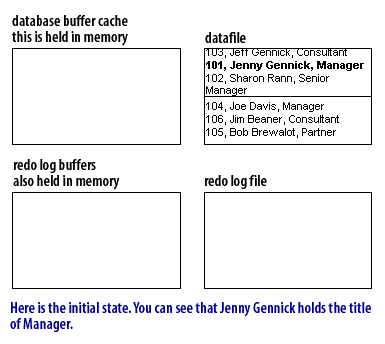
SYSTEM C:\ORAWIN95\DATABASE\SYS1ORCL.ORA

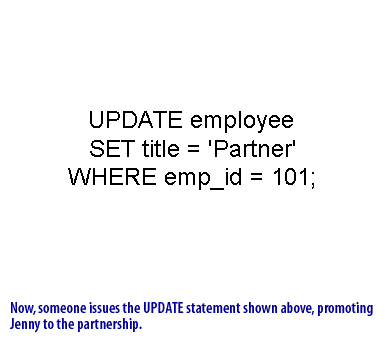
TEMPORARY\_DATA C:\ORAWIN95\DATABASE\TMP1ORCL.ORA

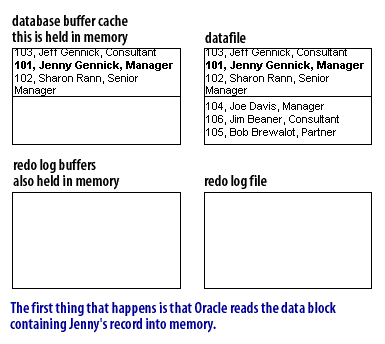
USER\_DATA C:\ORAWIN95\DATABASE\USR1ORCL.ORA

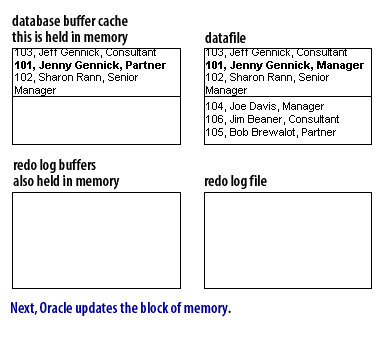
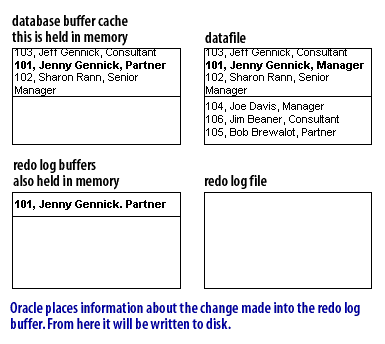
## Why redo logs are needed

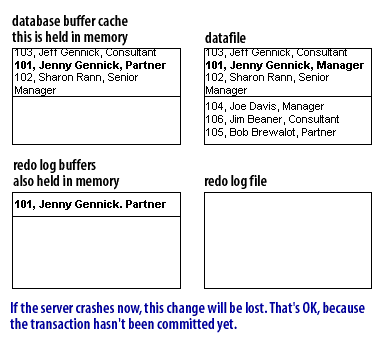
Another type of file that you will find as part of your database is the redo log file. Redo logs enable Oracle software to recover from a server crash, and they allow for up-to-the-minute recovery if a database needs to be restored from a backup. Whenever you issue a transaction that changes data in a database, the change is written both to the datafiles and to the redo log files, but not neccessarilly at the same time. Look through the following SlideShow to see how this happens:

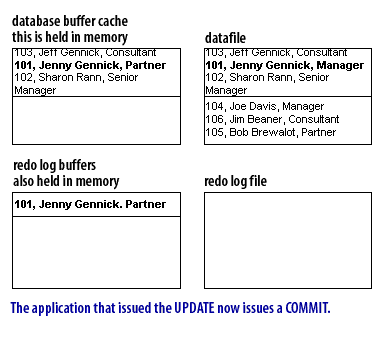


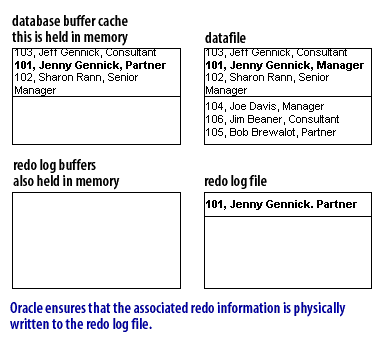


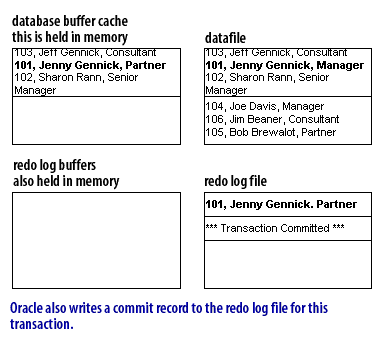


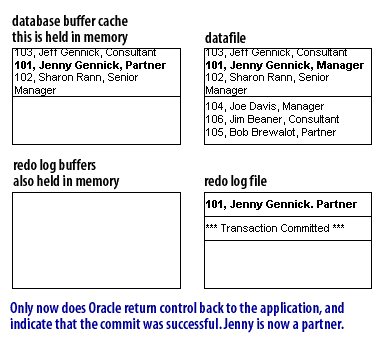
 

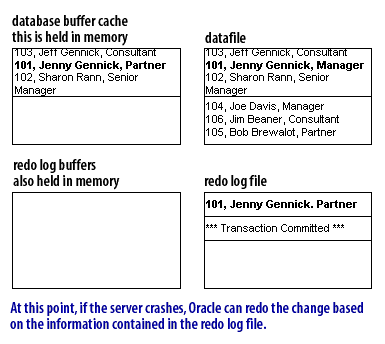


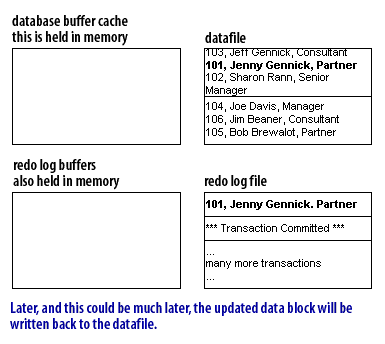












[**Redo log file Process**](https://www.relationaldbdesign.com/database-architecture/module3/redo-log-file-process.php)

### How redo logs work

As you can see, Oracle does not immediately write changes to the datafiles .

[**Find out why**](https://www.relationaldbdesign.com/database-architecture/module3/write-changes-immediately.php).  
However, when you commit a transaction, Oracle does guarantee that the changes for that transaction are written to the redo log, and that the commit or COMMIT has been recorded as well. If the server crashes, any changes in memory are lost. However, changes written to the redo log are not lost, because they are already on disk. When you restart the Oracle software, it checks the redo log, and reapplies (or redoes) to datafiles any committed transactions that were lost during the crash.  
Click on the following link to read about the [Redo Logs](https://www.relationaldbdesign.com/database-architecture/module3/redo-log-process.php) .

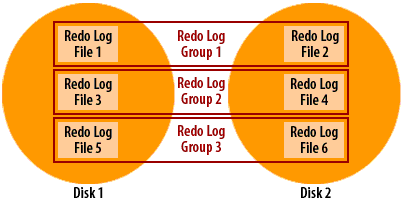
**Oracle recommendations**

Redo log files are critical to the process of recovering from a server crash. If you lose your redo log files during a server crash, perhaps due to a disk failure, you will not be able to recover the database. Because the redo log files are so critical, Oracle makes the following recommendations:

1. *Multiplex* your redo log files
2. Place *redo log members* on separate disks

Multiplexing your redo logs means to have the Oracle software write two or more copies of each log file. The following diagram illustrates a typical multiplexing situation:

Multiplexing your redo logs means to have the Oracle software write two or more copies of each log file. The following diagram illustrates a typical multiplexing situation:



Multiplexing redo log groups to have the Oracle software write 2 or more copies of each log file

### Multiplexing redo logs

Oracle uses the term redo log group to refer to a multiplexed set of redo log files. Oracle writes the same information to each file in a group. In the above example, redo log files 1 and 2 form one group and contain identical information. Oracle uses the term redo log member to refer to the individual files. You will learn the actual steps that make up the multiplexing process later in the course. Right now, you just need to understand why the multiplexing of the redo logs is important.

### Placing redo logs on separate disks

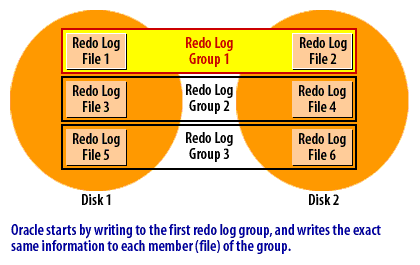
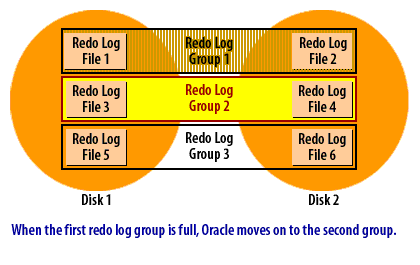
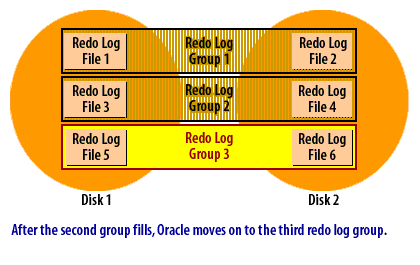
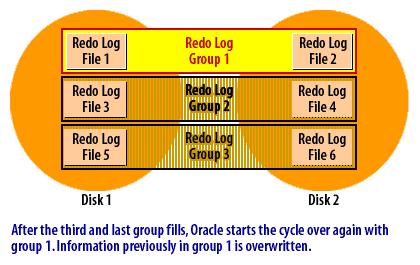
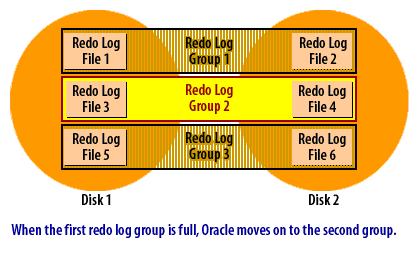
Oracle's second recommendation is to place members of a redo log group on separate disks, and this is critical. The whole reason for having two or more members in a group is so that you can recover if either of those disks fails. If all your redo log files are on one disk, and that disk fails, you have lost your database.

## Oracle cycles through Redo Logs

### Multiple redo log files

Oracle writes to redo log files in a cyclical fashion, first writing to one group, then another, and so forth, eventually coming around once again to the first group. We will examine how to specify these groups later in the course. The very first time that you create and start a database, Oracle will open the first group of redo log files. As you make changes to the database, Oracle writes a record of those changes to the redo logs. When the first group fills up, Oracle starts to write changes to the second group. This process continues for the life of the database. Flip through the following SlideShow to see this process in action.

## Cycling through Oracle redo logs

Oracle starts by writing to the first redo log group, and writes the exact same information to each member file of the group  
When the first redo log group is full, Oracle moves on to the second group.  
After the second group fills, Oracle moves on to the third redo log group.  
[**Oracle Database Administration**](https://amzn.to/2G91as3)  
After the third and last group fills, Oracle starts the cycle over again with group 1. Information previously in group1 is overwritten.  
When the first redo log group is full, Oracle moves on to the second group.

### Why multiple redo log files are needed

An Oracle database must always have a minimum of two redo log groups. The reason for this is the cyclical process that you just saw in the SlideShow. When one redo log group fills up, Oracle needs to be able to start writing another. Oracle cannot overwrite the redo log file that it just filled. At a minimum, you must have two groups, so that you can alternate back and forth between the two.

### The log switch

The point at which Oracle advances from writing one redo log group to writing the next is known as a log switch.  
When a log switch occurs, Oracle must ensure that the new redo log group is no longer needed for recovery. If any of the records in that redo log group reflect changes that still exist only in memory, then Oracle must write the affected data blocks to disk before the log switch can take place. Oracle keeps track of this for you and ensures that it happens.

[**Here is why Oracle does this**](https://www.relationaldbdesign.com/database-architecture/module3/checkpoint-control-file.php).  
If archive log mode is on, Oracle will also need to ensure that the next redo log group has been archived before proceeding with the log switch. This will be covered in more detail later in the course.

## Looking at Oracle redo log information

### Listing redo log groups

Two system views, V$LOG and V$LOGFILE, give you information about the redo log files for your database. The following example shows how you can query the V$LOG view to list the redo log groups in your database:

SVRMGR> SELECT group#, members, bytes, status

2> FROM v$log;

GROUP# MEMBERS BYTES STATUS

---------- ---------- ---------- ----------------

1 2 204800 CURRENT

2 2 204800 INACTIVE

3 2 204800 INACTIVE

3 rows selected.

In this example, there are three redo log groups. The redo log file sizes are 200K, or 204,800 bytes, and there are two members in each group. The status of group #1 is CURRENT, which means that Oracle is currently writing to the log files in that group.

### Finding the file names of the redo log files

You can use the V$LOGFILE view to find out the actual filenames of the log files. The following example shows how:

SVRMGR> SELECT group#, member

2> FROM v$logfile

3> ORDER BY group#;

GROUP# MEMBER

---------- --------------------------------------

1 C:\ORANT\DATABASE\LOG2ORCL.ORA

1 D:\ORANT\DATABASE\LOG2ORCL.ORA

2 C:\ORANT\DATABASE\LOG1ORCL.ORA

2 D:\ORANT\DATABASE\LOG1ORCL.ORA

3 C:\ORANT\DATABASE\LOG3ORCL.ORA

3 D:\ORANT\DATABASE\LOG3ORCL.ORA

6 rows selected.

There are six groups, with two members each, for a total of six redo log files. Notice that members of each group are on separate disks in order to provide redundancy, and protect the data in the event that one disk fails. Later in this course, when you start to create your own database, you will learn how to create these files, and specify which disks to use. Next, you will learn about archive log files.

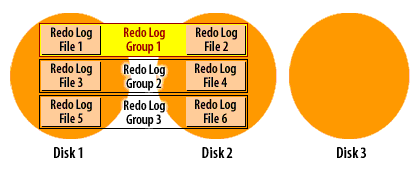
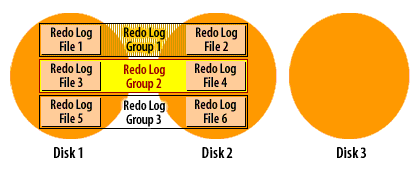
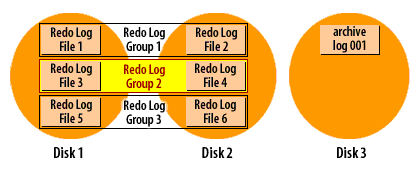
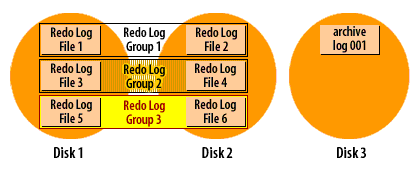
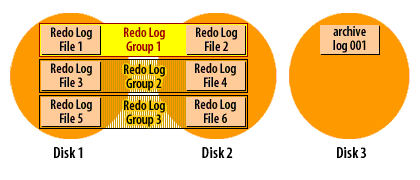
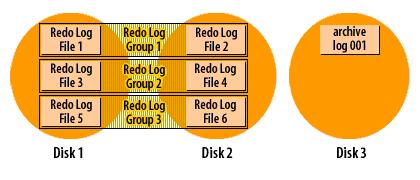
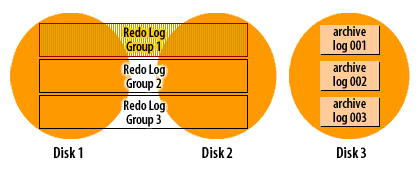
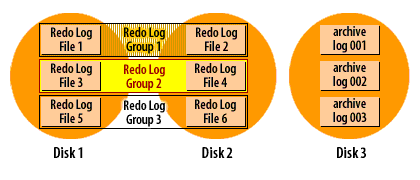
## Archived Redo Logs in Oracle

[**Generating Change Reports**](https://www.relationaldbdesign.com/database-architecture/module3/generating-change-reports.php)  
Explain how archived redo logs provide point-in-time recovery.  
You have just learned about how redo logs can be used to replay transactions lost during a system crash. They can also be used to replay transactions when a database has been restored from a backup. This enables you to restore a database from a backup, and recover it up to the very moment of failure, without losing any committed transactions. The key to doing this is to save all the redo logs generated since the most recent backup. You do this by running your database in archive log mode, which you will learn about later in this course.

### How archive log mode works

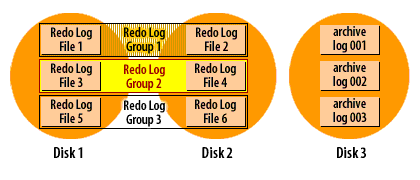
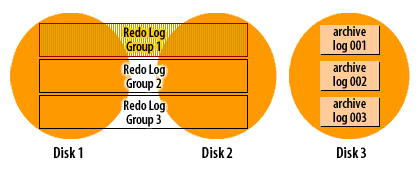
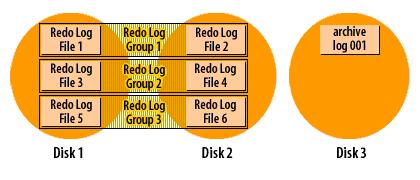
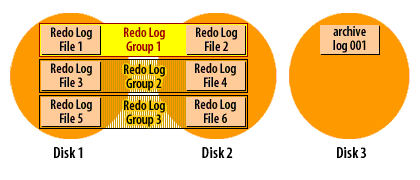
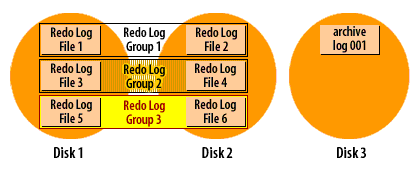
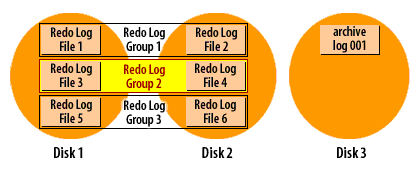
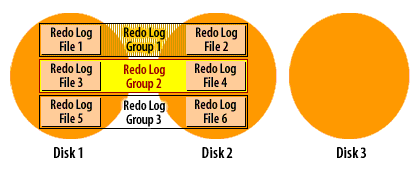
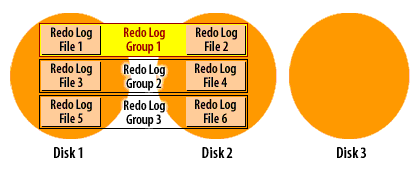
Recall that Oracle cycles through the redo log files in a circular fashion, reusing each file over and over again. When you run a database in archive log mode, Oracle makes a copy of each redo log file before it is reused. The following SlideShow demonstrates how it works:

## Oracle Redo log cycle in archive log mode

1) The redo logs are spread over disks 1 and 2. The archive log destination is disk 3. Right now, Oracle is writing to redo log group 1.  
2) A log switch has occurred, and Oracle is now writing to redo log group 2.  
3) At some point after the log switch, Oracle will copy one of the group 1 members to the archive log destination.  
4) Now Oracle has advanced to redo log group 3, and group 2 is available to be archived.  
[**Oracle Database Administration**](https://amzn.to/2G91as3)  
5) Oracle fills group 3, and rolls back around to group 1. The archiver has fallen a bit behind. Group 2 has not been archived yet.  
6) Oracle fills group 1, but cannot advance to group2 because it has not been archived yet. This is not good, because now Oracle has to wait.  
7) Finally, the archiver catches up a bit and group2 is archived.  


8) Oracle is now free to overwrite redo log group 2, so the log switch can occur. The process continues ad infinitum.

1. The redo logs are spread over disks 1 and 2.
2. A log switch has occurred, and Oracle is now writing to redo log group 2.
3. At some point after the log switch, Oracle will copy one of the group 1 members to the archive log destination.
4. Now Oracle has advanced to redo log group 3, and group 2 is available to be archived
5. Oracle fills group 3, and rolls back around to group 1.
6. Oracle fills group 1, but cannot advance to group2 because it has not been archived yet.
7. Finally, the archiver catches up a bit and group2 is archived.
8. Oracle is now free to overwrite redo log group 2, so the log switch can occur.



[Archive log mode](https://www.relationaldbdesign.com/database-architecture/module3/archive-log-mode.php)

**Naming of archive log files**

As you can see from the SlideShow, the redo log filenames do not carry over when a log file is archived. Instead, archive log files are sequentially numbered. You do have control over the format of the filename through the use of the LOG\_ARCHIVE\_FORMAT initialization parameter. You will learn about that in a later course in this series.

**Timing of the archiver process**

The copying of files is done asynchronously with respect to the log switches and Oracle only copies one member of each group to the archive log destination. When a log switch occurs, the archiver process is notified. The archiver will eventually copy the log file. As the DBA, you hope this will start right away, but the archiver may be busy with a previous file. The important point is that database users do not have to wait while files are archived. As long as the archiver gets caught up before Oracle needs to reuse that log file,the archiving process will run smoothly. There are a number of things that affect whether or not the archiver will be caught up at any given moment: disk I/O speed, disk contention, CPU utilization, and the transaction rate. A sudden burst of activity could cause the archiver to slow down for awhile. That is OK, as long as it can keep up with the overall throughput.

This archiving process is one more reason why you must have a minimum of two redo log groups in an Oracle database. In order for the archiver process to make a copy of a filled redo log file, that file must be closed long enough for the copy to take place. The only way to close a redo log file and still keep the database running, is to switch to another redo log group, and that's exactly what Oracle8i does.

**Oracle Database Architecture Conclusion**

The following is what was discussed in this module. An Oracle database consists of the following types of files:

1. *An initialization file:* Points to the control files, and contains a number of parameters affecting database operation.
2. *Control files:* Point to the other database files, and contains information crucial to the operation of the database.
3. *Datafiles:* Hold your table and index data.
4. *Log files:* Keep a record of changes that have been made, and transactions that have been committed, for use when recovering the database.
5. *Archived log files:* Are copies of the redo log files and enable point-in-time recovery of a database.

You have also learned about tablespaces, that they are a logical construct sitting between schema objects such as tables, and thephysical datafiles used to store those objects. Tablespaces provide you with flexibility in how your data is physically stored on disk.

Finally, you saw in detail how Oracle writes a record of database changes to the redo log. You saw that Oracle uses redo log files over and over, in a circular fashion, and you saw how Oracle archives those files when a database is running in archive log mode.

**Oracle Database Architecture Glossary**

This module introduced you to the following terms:

1. *archive log mode:*A mode of the database that enables the archiving of the online redo log.
2. *archive log file:*
3. *background processes:* A background process is a computer process that runs behind the scenes and without user intervention. Typical tasks for these processes include logging, system monitoring, scheduling, and user notification.
4. *checkpoint:* 1. A data structure that marks the checkpoint position, which is the SCN in the redo thread where instance recovery must begin. Checkpoints are recorded in the control file and each data file header, and are a crucial element of recovery. 2. The writing of dirty data blocks in the database buffer cache to disk. The database writer (DBW) process writes blocks to disk to synchronize the buffer cache with the data files.
5. *commit:* Action that ends a database transaction and makes permanent all changes performed in the transaction.
6. *controllers:* Usually refers to disk controllers. A disk controller is a component (usually a circuit board card) that manages one or more disks.
7. *database buffer cache:* The portion of the system global area (SGA) that holds copies of data blocks. All client processes concurrently connected to the database instance share access to the buffer cache.
8. *instance:* The term instance refers to a set of memory structures and background processes that operate against an Oracle database. Database users (programs that you run) communicate with an Oracle instance (background processes), and the instance does the actual work of reading and writing data to and from the database files. Using Oracle Parallel Server option, it is possible to have multiple instances simultaneously operating against a single database.
9. *log switch:* The point at which the log writer process (LGWR) stops writing to the active redo log file and switches to the next available redo log file. LGWR switches when either the active redo log file is filled with redo records or a switch is manually initiated.
10. *multiplex:* By multiplexing a control file on different disks, the database can achieve redundancy and thereby avoid a single point of failure.
11. *redo log group:* A group of redo log files that Oracle treats as a unit. Oracle writes the same information to each file, thus implementing a form of mirroring using software rather than hardware.
12. *redo log member:* One file in a group of redo log files.
13. *redo log file:* A file that contains part of a databases redo log
14. *system tablespace:* The tablespace Oracle uses for the data dictionary. Oracle treats this differently from other tablespaces. It is the one tablespace created by the CREATE DATABASE statement. Every database must have one, and it can never be taken offline.
15. *storage parameters:* Values that control the way in which Oracle allocates disk space for a given table, view, or other database object. Storage parameters allow you to control the size of an object's initial extent and subsequent extents, and the maximum number of extents.
16. *tablespace:* A tablespace is a storage location where the actual data underlying database objects can be kept. It provides a layer of abstraction between physical and logical data, and serves to allocate storage for all DBMS managed segments.

Now that you are familiar with the file types used in an Oracle database, we will turn our attention in the next module to the various Oracle server processes that operate on those files.

**Oracle Instance Architecture**

This module discusses the processes that make a running Oracle database work. After completing this module, you will be able to:

1. Explain the difference between a database and an instance
2. Identify the major background processes of an Oracle instance, and explain their purpose
3. Display a list of the Oracle background processes that are currently running on your server
4. Identify the instances running on your system based on a listing of process names

The first thing to understand is the difference between a database and an instance.

1. Instance versus database
2. Components of an instance
3. Creating the OFA file structure ($DBA, bdump, udump, pfile) will be discussed later in this module

**Database Instance Structure**

When an instance is started, Oracle Database allocates a memory area called the system global area (SGA) and starts one or more background processes. The SGA serves various purposes, including the following:

1. Maintaining internal data structures that are accessed by many processes and threads concurrently
2. Caching data blocks read from disk
3. Buffering redo data before writing it to the online redo log files
4. Storing SQL execution plans

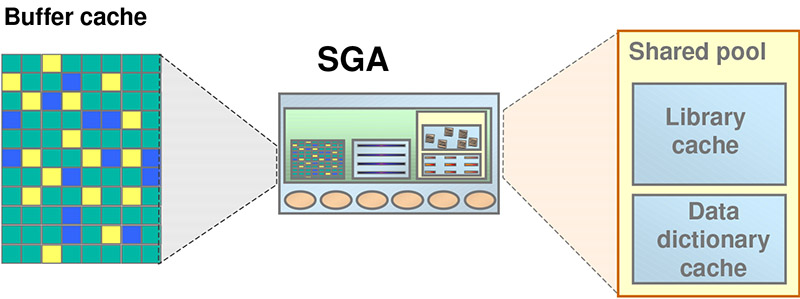
The SGA is shared by the Oracle processes, which include server processes and background processes, running on a single computer. The way in which Oracle processes are associated with the SGA is dependent on the operating system.

A database instance includes

1. background processes,
2. server processes, and
3. the process memory

allocated in these processes

The instance continues to function when server processes terminate.

1) Buffer cache, 2) SGA, and 3) Shared Pool  
The large shared memory segment, the SGA, contains the following major components:

1. The buffer cache, which is a cache of disk blocks, very similar to the file system cache found in most operating systems. Blocks are always read and written in sizes of the Oracle Block Size, which is defined, when the database is created, although tablespaces can be added with a different block size.
2. The shared pool, which contains two main components.
3. The library cache, which is a cache of SQL statements, etc.
4. The dictionary cache, which caches Oracle own internal information, such as information about users and tables.

## Difference between an instance and database in Oracle

The terms 1)instance and 2) database are often used interchangeably, or otherwise slightly abused, in spite of the fact that they each have distinct meanings. I think the reason for this lies with human nature, and our penchant for being somewhat lazy. It's very cumbersome to always say “I started an instance and opened the database.” That does not roll off the tongue too well. It's certainly a lot easier just to say “I started the database.” People who have used Oracle software for years, even those who work for Oracle, will tend to be somewhat lax in their use of these terms, and that's OK--so long as both sides understand what's being said. Still, there are times when precision is important. So make sure that you understand that a database is just the files, and that an instance is processes + memory.

## Overview of the Oracle Instance

An Oracle database server consists of an Oracle database and an Oracle instance. Every time a database is started, a system global area (SGA) is allocated, and Oracle background processes are started. The combination of the background processes and memory buffers is called an Oracle instance

## Oracle Database Architecture

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A server reliably manages a large amount of data in a multi-user environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance and a database server also prevents unauthorized access and provides efficient solutions for failure recovery.

The Oracle Database is the first database designed for

enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components and there is no need for peak workloads, since capacity can be easily added or reallocated from the resource pools as needed.  
A database consists of logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

Two key terms that you will hear used a lot once you start working with Oracle are instance and

database. These terms have very precise meanings in the Oracle world.

### Database

Oracle uses the term database[[1]](https://www.relationaldbdesign.com/database-architecture/module4/difference-between-instance-database.php#fn1) to refer to the collection of files that contain your data. This includes datafiles, redo logs, and control files. You learned about these files in the previous module.

### Instance

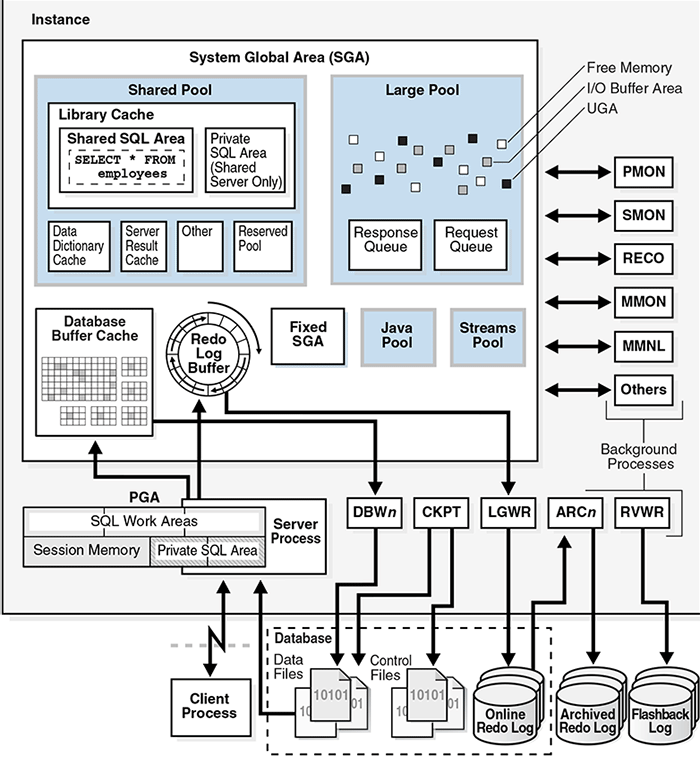
Oracle uses the term instance[[2]](https://www.relationaldbdesign.com/database-architecture/module4/difference-between-instance-database.php#fn2) to refer to the processes and memory structures that exist when an Oracle database is open and in use. These processes perform tasks such as writing changed data back to the datafiles, writing the redo log, and archiving filled log files.  
An analogy that I often use to illuminate this concept is to compare Oracle to Microsoft Word. An Oracle database is analogous to a Word document. It is a collection of files containing data. The Oracle instance could be compared to the Microsoft Word application program. In order to do something with your Word document, you need to start Word and open the document. The same applies to Oracle. To do anything with an Oracle database, you need to first start an instance. Once you have an instance running, you can open a database and perform the desired tasks.

## Oracle Database Architecture

A database server is the key to information management. In general, a server reliably manages a large amount of data in a multiuser environment so that users can concurrently access the same data. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

## Database and Instance

An Oracle database server consists of a database and at least one database instance (commonly referred to as simply an instance). Because an instance and a database are so closely connected, the term Oracle database is sometimes used to refer to both instance and database. In the strictest sense the terms have the following meanings:

Figure 1 - Shows a database and its instance. For each user connection to the instance, a client process runs the application

### Physical versus Logical Data

A database can be considered from both a 1) physical and 2) logical perspective.

Physical data is data viewable at the operating system level. For example, operating system utilities such as the Linux ls and ps can list database files and processes.  
Logical data such as a table is meaningful only for the database. A SQL statement can list the tables in an Oracle database, but an operating system utility cannot. The database has physical structures and logical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures. For example, renaming a physical database file does not rename the tables whose data is stored in this file.

[[1]](https://www.relationaldbdesign.com/database-architecture/module4/difference-between-instance-database.php#r1) Database: A database is a set of files, located on disk, that store data. These files can exist independently of a database instance.

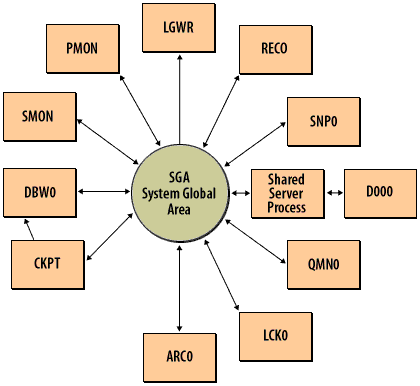
[[2]](https://www.relationaldbdesign.com/database-architecture/module4/difference-between-instance-database.php#r2) Database instance: An instance is a set of memory structures that manage database files. The instance consists of a shared memory area, called the system global area (SGA), and a set of background processes. An instance can exist independently of database files.

## Components that make up an Oracle Instance

The diagram below illustrates the processes and memory structures that make up an Oracle instance. When you move your mouse over a box, you will see a brief description of what each particular process does.  
The square boxes represent processes, while the circle in the center represents an area of memory that they all share.

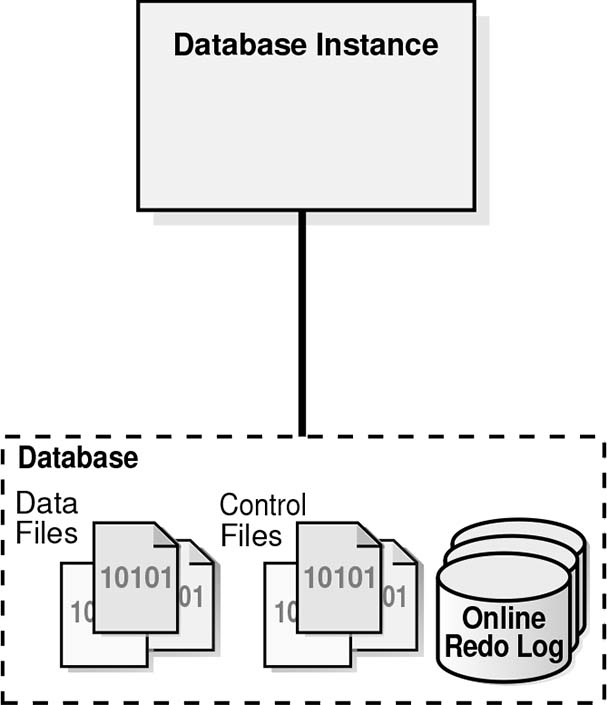
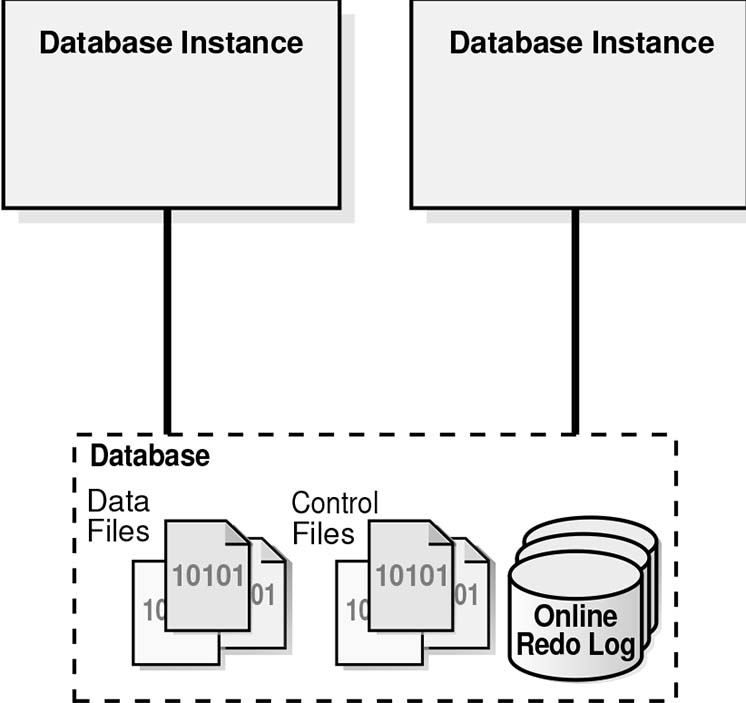
Boxes drawn in bold are processes that are mandatory, and that will always be running when a database is open. The other processes are optional, and whether or not they will exist depends on the specific mix of Oracle features that you are using.

**What makes up an Oracle Instance**

Oracle Instance Components: 1) SGA, 2) DBW0 3) LGWR 4) SMON 5) PMON 6) RECO 7) SNP0

1. *SGA - System Global Area:* A memory area containing the database buffer cache, the redo log buffer, frequently used SQL statements, and other items critical to the operation of a database.
2. *DBW0 - Database Writer*: Writes modified blocks back to the datafiles.
3. LGWR - Log Writer: Writes the database's redo log.
4. SMON - System Monitor: Performs crash recovery, coalesces free space, etc.
5. PMON - Process Monitor: Cleans up after user processes.
6. RECO - Recoverer: Resolves distributed transactions.
7. SNP0 - Snapshot: Runs the job queues used, among other things, for snapshot refreshes.
8. D000 - Dispatcher: Used with the multi-threaded server configuration.
9. Shared Server Process - Shared Server Process: Used with the multi-threaded server configuration, and not considered part of the instance.
10. QMN0 - Queue Monitor: Used with Oracle Advanced Queueing.
11. LCK0 - Lock: Used with parallel server for inter-instance locking.
12. CKPT - Checkpoint: Initiates and controls the checkpoint process.
13. ARC0 - Archiver: Copies filled redo log files to the archive log destination.

**Database Instance Configurations**

Oracle Database runs in either a single-instance configuration or an Oracle Real Application Clusters (Oracle RAC) configuration. These configurations are mutually exclusive. In a single-instance configuration, a one-to-one relationship exists between the database and a database instance as shown in figure 4-3.1.Figure 4-3.1: Single Instance DatabaseIn Oracle RAC, a one-to-many relationship exists between the database and database instances. The following figure 4-3.2 shows database instance configuration for RAC.Figure 4-3.2: Oracle RAC Database  
Whether in a single-instance or Oracle RAC configuration, a database instance is associated with only one database at a time. You can start a database instance and mount (associate the instance with) one database, but not mount two databases simultaneously with the same instance

**Simultaneous processes**

You will notice that several of the process names end with one or more zeros. DBW0 is one example. An Oracle instance can be configured to have several such processes running simultaneously. For example, if your database experiences a high volume of updates,, you might configure the instance to run with two database writers, which would be named DBW0 and DBW1. This generally only makes sense when you have multiple CPUs to run those processes.

The procedures for displaying a list of running processes are different, and depend on whether you are using

1. [**UNIX**](https://www.relationaldbdesign.com/database-architecture/module4/listing-unix-system-processes.php) or
2. [**Windows**](https://www.relationaldbdesign.com/database-architecture/module4/oracle-processes-under-windows.php).

When an instance is started, Oracle Database allocates a memory area and starts background processes. The memory area stores information such as the following:

1. Program code
2. Information about each connected session, even if it is not currently active
3. Information needed during program execution, for example, the current state of a query from which rows are being fetched
4. Information such as lock data that is shared and communicated among processes
5. Cached data, such as data blocks and redo records, that also exists on disk

**Asynchronous Process Coordination**

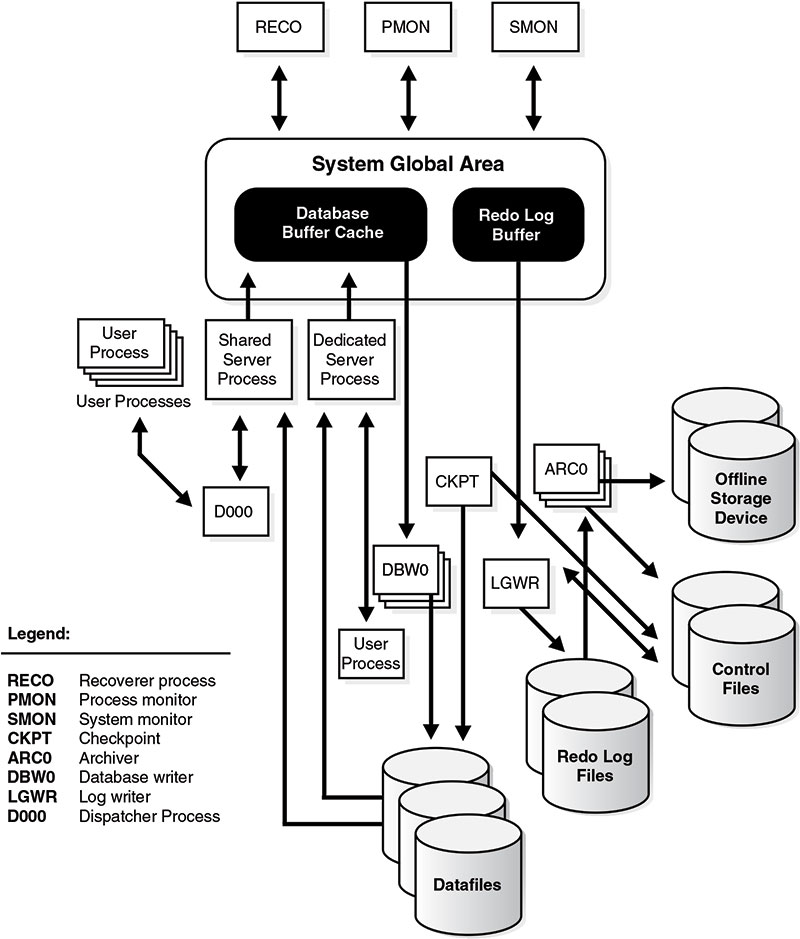
**Asynchronous processes run independently**

The Oracle background processes all run *asynchronously*. Each process executes as independently as possible. For example, at the same time that the log writer is busy writing records to the redo log files, the database writer is busy writing changed buffers to disk, and the archiver is copying full redo log files to the archive log destination. Occasionally, one process does need to wait for another, but that is not usually the case. The Database Writer (DBW0), for example, will not write a modified

*buffer* back to a datafile until the Log Writer (LGWR) process has logged all the changes for that buffer.

**Advantages of asynchronous processes**

This asynchronous behavior is part of what makes Oracle perform so well. A program can issue a query that changes a buffer without having to wait for that change to be written to disk. If the database is busy, the Database Writer doesn't need to write anything as long as enough space remains in the database buffer cache to hold the new data being read. The Log Writer process focuses only on writing the redo log files, and hopefully never needs to wait on the archiver to copy them.  
Figure 4-4 illustrates how each background process interacts with the different parts of an Oracle database.

Figure 4-4 Background Processes of a Multiple Process Oracle Instance

**Oracle Background Processes**

An Oracle database uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of the computers that constitute the database system. Processes are jobs that work in the memory of these computers. The architectural features discussed in this section enable the Oracle database to support:

1. Many users concurrently accessing a single database
2. The high performance required by concurrent multiuser, multiapplication database systems

Oracle creates a set of background processes for each instance. The background processes consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. They asynchronously perform I/O and monitor other Oracle process to provide increased parallelism for better performance and reliability.

**Mandatory Background Processes**

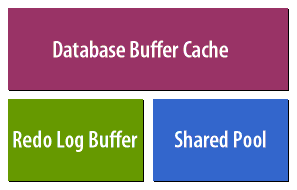
The mandatory background processes are present in all typical database configurations. These processes run by default in a database instance started with a minimally configured initialization parameter file. These are some of the mandatory background processes:

1. Process Monitor Process (PMON)
2. System Monitor Process (SMON)
3. Database Writer Process (DBWn)
4. Log Writer Process (LGWR)
5. Checkpoint Process (CKPT)
6. Manageability Monitor Processes (MMON and MMNL)
7. Recoverer Process (RECO)

**System Global Area (SGA)**

**Function of the SGA**

The *System Global Area*, or SGA, is a shared memory structure that is the focal point of activity for an Oracle instance. The SGA contains both data and control information for an instance, and is referenced in one way or another by virtually all of the background processes. Move your mouse over the image below to see the major components of the SGA.

Database Buffer Cache, Redo Log Buffer, Shared Pool

1. (Database buffer cache): Holds data blocks that have been read from datafiles.
2. (Redo Log Buffer): Holds redo log records prior to their being written to the log files.
3. (Shared Pool): Contains parsed versions of SQL statements and other information.

[Database Buffer Cache](https://www.relationaldbdesign.com/database-architecture/module4/database-buffer-cache.php)

**Benefits of the SGA**

Oracle uses this shared memory structure because it is a much more efficient way to share data between processes than other mechanisms.  
The size of the SGA can be significant, easily reaching several hundred megabytes for a large database. The SGA has a significant impact on database performance, and it's important to understand the various components so that you can size them properly. The next module covers the SGA in detail.

The (SGA) System Global Area is a group of shared memory areas that are dedicated to an *Oracle instance*, which is an instance is your database programs and RAM.

All Oracle processes use the SGA to hold information and the SGA is used to store incoming data.

1. the data buffers as defined by the db\_cache\_size parameter), and
2. internal control information that is needed by the database.

You control the amount of memory to be allocated to the SGA by setting the Oracle *initialization parameters*. These may include

1. db\_cache\_size,
2. shared\_pool\_size and
3. log\_buffer.

In Oracle Database 10g you only need to define two parameters, namely

1. sga\_target and
2. sga\_max\_size

to configure your SGA. If these parameters are configured, Oracle will calculate how much memory to allocate to the different areas of the SGA using a feature called (AMM) Automatic Memory Management . As you progress you may want to manually allocate memory to each individual area of the SGA using the *initialization parameters*.

**Purpose of SMON Process**

The System Monitor (SMON, pronounced “ess-mahn”) process is responsible for:

1. *Instance crash recovery*
2. Cleaning up *temporary segments*
3. Coalescing free spaces

**System Monitor Process (SMON)**

The system monitor process (SMON) performs recovery, if necessary, at instance startup. SMON is also responsible for cleaning up temporary segments that are no longer in use and for *coalescing contiguous free extents* within dictionary managed tablespaces. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online. SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it. With Real Application Clusters, the SMON process of one instance can perform instance recovery for a failed CPU or instance.

The system monitor process (SMON) is in charge of a variety of system-level cleanup duties. The duties assigned to SMON include:

1. Performing instance recovery, if necessary, at instance startup. In an Oracle RAC database, the SMON process of one database instance can perform instance recovery for a failed instance.
2. Recovering terminated transactions that were skipped during instance recovery because of file-read or tablespace offline errors. SMON recovers the transactions when the tablespace or file is brought back online.
3. Cleaning up unused temporary segments. For example, Oracle Database allocates extents when creating an index. If the operation fails, then SMON cleans up the temporary space.
4. Coalescing contiguous free extents within dictionary-managed tablespaces.

SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

**Crash recovery**

Remember that Oracle buffers data in the SGA, and does not necessarily write out changes to the datafiles immediately after they occur. If the instance crashes, any changes buffered in memory are lost. Whenever you start an Oracle instance, SMON checks to see if it has previously crashed. If it has, then SMON reads the redo log files, and applies any needed changes to the datafiles.

**Cleaning up temporary spaces**

Temporary segments are often used for sorting. When a sort finishes, SMON takes care of deallocating any segments used for that sort.

**Coalescing free space**

Coalescing free space refers to the practice of taking two adjacent areas of free space with a datafile and converting them into one larger area of free space. This procedure is repeated over and over in order to minimize fragmentation in the files.

**Oracle Process Monitor (PMON)**

The Process Monitor (PMON) process does the same sort of thing for user processes that SMON does for the instance--it takes care of various cleanup operations. Whenever a user process is abnormally terminated, PMON does the following:

1. Releases any locks that may have been held by the user process
2. Removes the process ID from the list of active processes
3. Rolls back any open transactions for the process

If the *multi-threaded server* option is being used, PMON will also periodically check to be sure that the necessary dispatcher processes are running.

**(PMON)Process Monitor**

The process monitor (PMON) performs process recovery when a user process fails. PMON is responsible for

1. cleaning up the database buffer cache and
2. freeing resources that the user process was using.

For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes. PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener.  
If a user process terminates abnormally, PMON is responsible for cleaning up any of the resources left behind (such as memory) and for releasing any locks held by the failed process.  
Like SMON, PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

## DBW0) Database Writer

The Database Writer (DBW0) process has one purpose in life: to write modified data back to the datafiles.

### Changed data in dirty buffers

Oracle holds the most recently read data from disk in a memory area known as the database buffer cache. Whenever any of that data is changed, the buffers with the changed data are marked as dirty. The database writer will periodically check for these dirty buffers, and write them back to the datafiles. The following SlideShow illustrates this process:

**Enhanced v$process view**

*RAM memory monitoring* is implemented in Oracle by enhancing the v$process view. The new columns in the

v$process view allow you to show details about the (PGA) program global area regions for all current Oracle processes. The PGA is a dedicated area of RAM memory used by individual processes to perform RAM intensive functions, such as sorting.  
The three new columns in the v$process view include

1. pga\_used\_memory,
2. pga\_allocated\_memory, and
3. pga\_max\_memory.

After analyzing these metrics, you can

1. see the actual RAM utilization for individual background processes within the Oracle environment and
2. also look at the RAM demands of individual connections to the database.

To illustrate, consider the following query:

col c1 heading 'Program|Name' format a30

col c2 heading 'PGA|Used|Memory' format 999,999,999

col c3 heading 'PGA|Allocated|Memory' format 999,999,999

col c4 heading 'PGA|Maximum|Memory' format 999,999,999

select

program c1,

pga\_used\_mem c2,

pga\_alloc\_mem c3,

pga\_max\_mem c4

from

v$process

order by

c4 desc;

**Figure 1**

PROGRAM PGA\_USED\_MEM PGA\_ALLOC\_MEM PGA\_MAX\_MEM

------------------------- ------------ ------------- -----------

oracle@john (PMON) 120,463 234,291 234,291

oracle@john (DBW0) 1,307,179 1,817,295 1,817,295

oracle@john (LGWR) 4,343,655 4,849,203 4,849,203

oracle@john (CKPT) 194,999 332,583 332,583

oracle@john (SMON) 179,923 775,311 775,323

oracle@john (RECO) 129,719 242,803 242,803

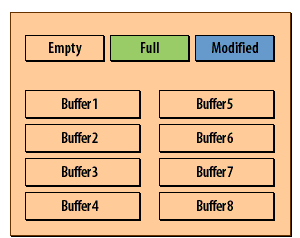
oracle@john (TNS V1-V3) 1,400,543 1,540,627 1,540,915

oracle@john (P000) 299,599 373,791 635,959

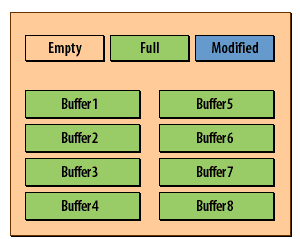
oracle@john (P001) 299,599 373,791 636,007

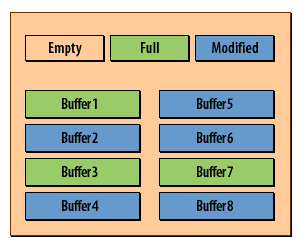
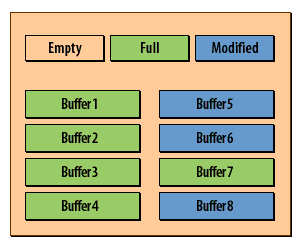
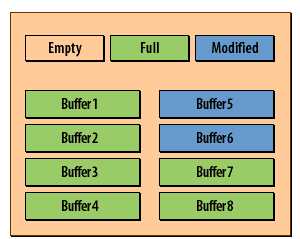
oracle@john (TNS V1-V3) 1,400,543 1,540,627 1,540,915

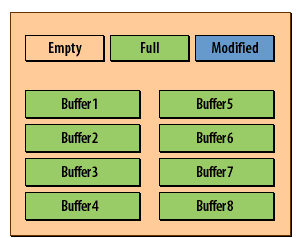
oracle@john (TNS V1-V3) 22,341 1,716,253 3,625,241

This example provides insight into the behavior of the Oracle database engine. One can see that the (LGWR) log writer process is the highest consumer of PGA RAM memory, which makes sense because the Oracle Log Writer process must transfer redo log images from the Log Buffer (in RAM memory) to the online redo log filesystem. You can also see high RAM memory utilization for the **(DBW0) Database Writer** process. The asynchronous I/O processes used by Oracle make extensive use of RAM memory resources to ensure that all database changes are successfully written to the database.  


1) When a database first starts, all the database buffers are empty.



2) As queries are executed, the cache begins to fill up. Blocks are read from disk and placed in the buffers.  
3) UPDATE, INSERT, and DELETE statements cause the data in some blocks to change.  
4) The database writer periodically checks for modified blocks and writes them back to disk.  
5) Modified blocks are written to disk, but also kept in memory in case they are needed again.

6) If database activity slows, the database writer will catch up. Here the buffers are full, but all changes have been written.

[**Oracle Database Writer**](https://www.relationaldbdesign.com/database-architecture/module4/oracle-database-writer.php)  
An instance may have [**up to ten**](https://www.relationaldbdesign.com/database-architecture/module4/multiple-database-writers.php) database writer processes.  
If an Oracle user process searches the threshold limit of buffers without finding a free buffer, the process stops searching the LRU list[[1]](https://www.relationaldbdesign.com/database-architecture/module4/dbw0-oracle-database-writer.php#fn1) and signals the DBW0 background process to write some of the dirty buffers to disk.

## The following facts hold for Oracle 12c

### Consider Multiple Database Writer (DBWR) Processes or I/O Slaves

Configuring multiple database writer processes, or using I/O slaves, is useful when the transaction rates are high or when the buffer cache size is so large that a single DBWn process cannot keep up with the load.

### DB\_WRITER\_PROCESSES

The DB\_WRITER\_PROCESSES initialization parameter lets you configure multiple database writer processes (from DBW0 to DBW9 and from DBWa to DBWj). Configuring multiple DBWR processes distributes the work required to identify buffers to be written, and it also distributes the I/O load over these processes. Multiple db writer processes are highly recommended for systems with multiple CPUs (at least one db writer for every 8 CPUs) or multiple processor groups (at least as many db writers as processor groups). Based upon the number of CPUs and the number of processor groups, Oracle Database either selects an appropriate default setting for DB\_WRITER\_PROCESSES or adjusts a user-specified setting.

### DBWR\_IO\_SLAVES

If it is not practical to use multiple DBWR processes, then Oracle Database provides a facility whereby the I/O load can be distributed over multiple slave processes. The DBWR process is the only process that scans the buffer cache LRU list for blocks to be written out. However, the I/O for those blocks is performed by the I/O slaves. The number of I/O slaves is determined by the parameter DBWR\_IO\_SLAVES.

DBWR\_IO\_SLAVES is intended for scenarios where you cannot use multiple DB\_WRITER\_PROCESSES (for example, where you have a single CPU). I/O slaves are also useful when asynchronous I/O is not available, because the multiple I/O slaves simulate nonblocking, asynchronous requests by freeing DBWR to continue identifying blocks in the cache to be written. Asynchronous I/O at the operating system level, if you have it, is generally preferred. DBWR I/O slaves are allocated immediately following database open when the first I/O request is made. The DBWR continues to perform all of the DBWR-related work, apart from performing I/O. I/O slaves simply perform the I/O on behalf of DBWR. The writing of the batch is parallelized between the I/O slaves.

### LRU Algorithm and Full Table Scans

[[1]](https://www.relationaldbdesign.com/database-architecture/module4/dbw0-oracle-database-writer.php#r1) When the user process is performing a full table scan, it reads the blocks of the table into buffers and puts them on the LRU end (instead of the MRU end) of the LRU list. This is because a fully scanned table usually is needed only briefly, so the blocks should be moved out quickly to leave more frequently used blocks in the cache. You can control this default behavior of blocks involved in table scans on a table-by-table basis. To specify that blocks of the table are to be placed at the MRU end of the list during a full table scan, use the CACHE clause when creating or altering a table or cluster. You can specify this behavior for small lookup tables or large static historical tables to avoid I/O on subsequent accesses of the table.

**Oracle Database Writing Time Period**

Identify the situations that trigger the DBW0 process to write data.  
Three situations will cause the database writer to write dirty buffers back to disk:

1. Insufficient buffer space for data that needs to be read
2. Advancement of the *checkpoint*
3. The number of dirty blocks exceeds a predefined target

The next three lessons talk about each of these triggering situations in more detail.

**Database Writer Process (DBWn)**

The database writer process (DBWn) writes the contents of database buffers to data files. DBWn processes write modified buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9 and DBWa through DBWj) to improve write performance if your system modifies data heavily. These additional DBWn processes are not useful on uniprocessor systems. The DBWn process writes dirty buffers to disk under the following conditions:

1. When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers, it signals DBWn to write. DBWn writes dirty buffers to disk asynchronously if possible while performing other processing.
2. DBWn periodically writes buffers to advance the checkpoint, which is the position in the redo thread from which instance recovery begins.  
   The log position of the checkpoint is determined by the oldest dirty buffer in the buffer cache.

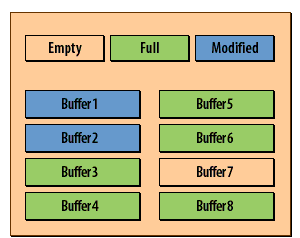
In many cases the blocks that DBWn writes are scattered throughout the disk. Thus, the writes tend to be slower than the sequential writes performed by LGWR. DBWn performs multiblock writes when possible to improve efficiency. The number of blocks written in a multiblock write varies by operating system.

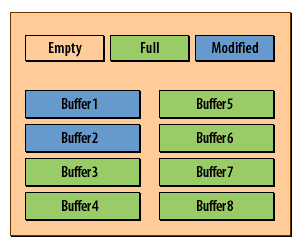
**DBWR - Database Writer**

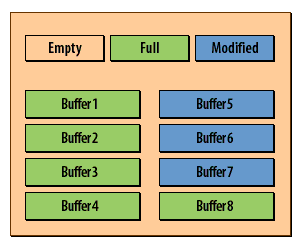
DBWR - Database Writer or Dirty Buffer Writer process is responsible for writing dirty buffers from the database block cache to the database data files. Generally, DBWR only writes blocks back to the data files on commit, or when the cache is full and space has to be made for more blocks. The possible multiple DBWR processes in RAC must be coordinated through the locking and global cache processes to ensure efficient processing is accomplished.

**How Oracle finds Space to write to new Data**

Reading new data from disk is one of the things that can indirectly trigger the database writer to write *dirty buffers* back to disk. When Oracle reads a block of data from the disk, it must place that data somewhere in the database buffer cache. If it can find an empty buffer, it uses that. Otherwise, it looks for an old buffer that has not been changed. If it finds one, it overwrites it with the new data. After checking a certain number of old buffers, if an unchanged one has not been found, the DBWR process will be asked to start writing some data back to disk in order to free up space for the new data. The following SlideShow demonstrates how this works.







1. In the diagram above, all the buffers except one are full
2. Oracle reads a block from disk, Oracle knows to place it in the empty buffer, causing it to become full.
3. If Oracle reads another block, it will need to overwrite one of the unchanged buffers.
4. It will check one buffer
5. It will check another buffer
6. It will check the next buffer until it finds an unmodified buffer that it can overwrite.
7. If many buffers are modified, Oracle may not find an unmodified one right away.
8. The database writer will be called upon to hurry up and write some modified blocks back to disk.
9. This increases the number of unmodified blocks
10. Oracle will use one to hold the block that was just read

[**Writing Out Buffers**](https://www.relationaldbdesign.com/database-architecture/module4/writing-out-buffers.php)

**Sequence for using Buffers**

When Oracle is checking for unmodified buffers it always starts with the *least recently used* buffer and works its way forward to the *most recently used*.  
Similarly when writing data, Oracle will write the [**least recently used**](https://www.relationaldbdesign.com/database-architecture/module4/least-recently-used.php) blocks first.

**Organization of the Database Buffer Cache**

The buffers in the cache are organized in two lists:

1. the write list and
2. the least recently used (LRU) list.

The write list holds dirty buffers, which contain data that has been modified but has not yet been written to disk. The LRU list holds free buffers, pinned buffers, and dirty buffers that have not yet been moved to the write list. Free buffers do not contain any useful data and are available for use. Pinned buffers are currently being accessed. When an Oracle process accesses a buffer, the process moves the buffer to the most recently used (MRU) end of the LRU list.  
As more buffers are continually moved to the MRU end of the LRU list, dirty buffers age toward the LRU end of the LRU list. The first time an Oracle user process requires a particular piece of data, it searches for the data in the *database buffer cache*. If the process finds the data already in the cache (a cache hit), it can read the data directly from memory.  
If the process cannot find the data in the cache (a cache miss), it must copy the data block from a datafile on disk into a buffer in the cache before accessing the data. Accessing data through a cache hit is faster than data access through a cache miss.  
Before reading a data block into the cache, the process must first find a free buffer. The process searches the LRU list, starting at the least recently used end of the list. The process searches either until it finds a free buffer or until it has searched the threshold limit of buffers. If the user process finds a dirty buffer as it searches the LRU list, it moves that buffer to the write list and continues to search. When the process finds a free buffer, it reads the data block from disk into the buffer and moves the buffer to the MRU end of the LRU list. If an Oracle user process searches the threshold limit of buffers without finding a free buffer, the process stops searching the LRU list and signals the DBW0 background process to write some of the dirty buffers to disk.

**Minimizing gaps between Redo logs and Data Files**

Checkpoint advancing can also trigger the database writer to write dirty blocks back to disk. This minimizes the number of changes that must be redone in the event the system crashes or the Oracle instance fails. Recall that each change to a database has a number assigned to it, and that this number corresponds to the redo log record written for that change. To prevent any inordinately large gaps between content in the redo logs and content in the datafiles, Oracle periodically writes the changes back to the datafiles. This is referred to as *checkpointing*.

**Writing frequently-changed blocks to disk**

The database buffer cache is part of the reason that checkpointing needs to occur. The buffer cache exists to keep the most frequently accessed data blocks in memory. Blocks that are changed a lot will tend to stay at the most recently used end of the LRU list. Because of this, they might never be forced back to disk by incoming data, as the previous lesson describes. Instead, they might sit in memory forever. You could make a million changes to them, and none of those changes would be written to disk. Obviously, that is not a good situation. Enter checkpointing. Oracle periodically takes a system change number, and ensures that all changes up through and including that change are written to disk. By default, a checkpoint occurs every time a redo log switch occurs.

**Overview of Checkpoints**

A checkpoint is a crucial mechanism in consistent database shutdowns, instance recovery, and Oracle Database operation generally. The term checkpoint has the following related meanings:

1. A data structure that indicates the checkpoint position, which is the SCN in the redo stream where instance recovery must begin The checkpoint position is determined by the oldest dirty buffer in the database buffer cache. The checkpoint position acts as a pointer to the redo stream and is stored in the control file and in each data file header.
2. The writing of modified database buffers in the database buffer cache to disk

**Redo Log Files**

Every Oracle database has a set of two or more redo log files. The set of redo log files is collectively known as the redo log for the database. A redo log is made up of redo entries (also called redo records). The primary function of the redo log is to record all changes made to data. If a failure prevents modified data from being permanently written to the datafiles, then the changes can be obtained from the redo log, so work is never lost. To protect against a failure involving the redo log itself, Oracle allows a multiplexed redo log so that two or more copies of the redo log can be maintained on different disks.

The information in a redo log file is used only to recover the database from a system or media failure that prevents database data from being written to the datafiles. For example, if an unexpected power outage terminates database operation, then data in memory cannot be written to the datafiles, and the data is lost. However, lost data can be recovered when the database is opened, after power is restored. By applying the information in the most recent redo log files to the database datafiles, Oracle restores the database to the time at which the power failure occurred. The process of applying the redo log during a recovery operation is called rolling forward.

## Limiting the Number of Dirty Buffers

### How to limit dirty buffers

You can use the DB\_BLOCK\_MAX\_DIRTY\_TARGET parameter to place an upper limit on the number of dirty buffers in the database buffer cache. You set this parameter in your initialization file, like this:

db\_block\_max\_dirty\_target=200

The example above limits the maximum number of dirty blocks to 200. The default behavior is to have no upper limit at all.

In this context, blocks and buffers are interchangeable. Strictly speaking, a buffer is an area in memory, a block is a piece of data from disk. However, Oracle always sizes the buffers in the buffer cache to match the block size. 200 dirty blocks is equal to 200 dirty buffers.

### Why limit dirty buffers?

The number of dirty buffers has a direct impact on the amount of time needed for crash recovery.  
That is because when the server crashes, it's the buffered data that you lose. When you restart the instance, those lost changes need to be reconstructed based on information in the redo logs. The more dirty buffers that are lost, the longer recovery will take.

### Recovery time versus performance

If you are in a time-sensitive environment, you might use the DB\_BLOCK\_MAX\_DIRTY\_TARGET parameter to keep the number of dirty blocks, and thus the recovery time, low. Of course this comes at a cost. Performance might suffer somewhat during periods of high activity, because Oracle will occasionally need to wait for the database writer to lower the dirty block count (by writing buffers back to the datafiles).

## DB\_WRITER\_PROCESSES

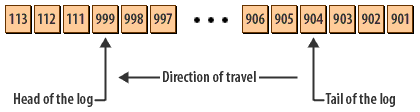
|  |  |
| --- | --- |
| Property | Description |
| Parameter type | Integer |
| Default value | 1 or CPU\_COUNT / 8, whichever is greater |
| Modifiable | No |
| Range of values | 1 to 20 |
| Basic | No |

DB\_WRITER\_PROCESSES is useful for systems that modify data heavily. It specifies the initial number of database writer processes for an instance.

**Log Writer (LGWR)**

Describe how the Log Writer writes out redo log records.  
Every Oracle instance will have a Log Writer process (LGWR) that is responsible for writing out

*redo log entries* to the redo log files. Unlike the database writer, an instance may have only one log writer.  
The Log Writer writes redo log entries from an area in the *SGA* known as the *redo log buffer*. The redo log buffer is a circular buffer, and conceptually looks like this:



1. Head of the log: Points to where new redo entries are added
2. Tail of the log: Points to the next redo log entry that needs to be written to disk
3. 113,112,111: Old redo log entries, long since written to disk
4. 903,902,901: Redo log entries that have just recently been written to disk
5. 998,997...906,905: Redo log entries that are waiting to be written to disk
6. 904: This will be the next redo log entry written to the redo log files.
7. 999: This is the most recently added redo log entry.

[**Oracle Logs**](https://www.relationaldbdesign.com/database-architecture/module4/log-writer-direction.php)

As changes are made to the database, redo log entries are added to the head of the log, and the head pointer is advanced. The LGWR process is constantly working from the other end of the log, the tail, and constantly writes redo log entries to disk. The next lesson explains this in more detail.

**Log Writer Process (LGWR)**

The log writer process (LGWR) is responsible for redo log buffer management.writing the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

Werbung

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy. LGWR writes one contiguous portion of the buffer to disk. LGWR writes:

1. A commit record when a user process commits a transaction
2. Redo log buffers
   1. Every three seconds
   2. When the redo log buffer is one-third full
   3. When a DBWn process writes modified buffers to disk, if necessary

LGWR writes synchronously to the active mirrored group of redo log files. If one of the files in the group is damaged or unavailable, LGWR continues writing to other files in the group and logs an error in the LGWR trace file and in the system alert log. If all files in a group are damaged, or the group is unavailable because it has not been archived, LGWR cannot continue to function. When a user issues a COMMIT statement, LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries.

Werbung

The corresponding changes to data blocks are deferred until it is more efficient to write them. This is called a fast commit mechanism. The atomic write of the redo entry containing the transaction's commit record is the single event that determines the transaction has committed. Oracle returns a success code to the committing transaction, although the data buffers have not yet been written to disk.

**Placing redo log entries, Writing to redo log files**

Oracle maintains two important pointers that you should be aware of.

1. One points to the *tail of the log* and
2. the other points to the *head of the log*.

As changes are made to a database, new redo log entries are always placed at the head of the log. The **Log Writer process**, on the other hand, always looks at the tail of the log. The Log Writer's job is to continually check the tail of the log to see if there are any new redo log records, and then to write those records to the redo log files as quickly as possible. The following SlideShow illustrates this process:

## Head and tail of the log

### Viewing Linux Files using tail command

Many times your only interest is in seeing the end of a file. To view just the last handful of lines in a file, use the tail command.

$ tail required\_packages.txt

elfutils-libelf-devel-0.97.1-5.i386.rpm

glibc-headers-2.3.4-2.41.i386.rpm

glibc-kernheaders-2.4-9.1.103.EL.i386.rpm will be required as a

prerequisite

glibc-devel-2.3.4-2.41.i386.rpm

gcc-3.4.6-10.i386.rpm

libstdc++-devel-3.4.6-10.i386.rpm

gcc-c++-3.4.6-10.i386.rpm

libaio-devel-0.3.105-2.i386.rpm

sysstat-5.0.5-19.el4.i386.rpm

unixODBC-devel-2.2.11-1.RHEL4.1.i386.rpm

By default, tail displays the last 10 lines of a file. This is a great way to look at recent entries in Oracle database alert logs.

If more than 10 lines need to be seen, specify a number of lines as an option. For example, tail -15 required\_packages.txt will show the last 15 lines of the file required\_packages.txt.  
When a log file is being actively written to, like with an alert log, displaying new lines on the screen as they are written to the file may be preferred. To do that, add the -f option to the tail command like this: tail -f alert\_TEST.log. This allows the monitoring of a log in near real time, though sometimes lines may appear too quickly to read.

### Viewing Linux Files using head command

$ head required\_packages.txt

REHEL4, OEL4:

Refer to Note 880211.1

binutils-2.15.92.0.2-25

compat-libstdc++-33-3.2.3-47.3

elfutils-libelf-0.97.1-5

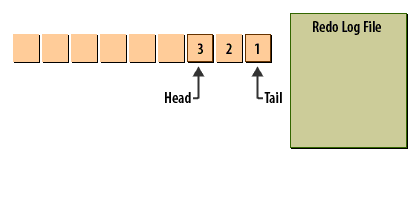
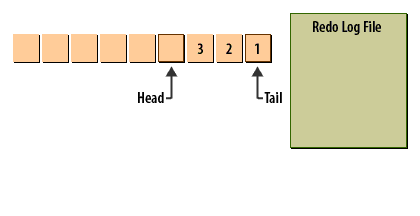
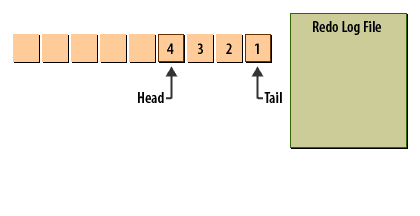
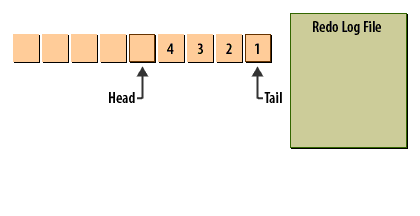
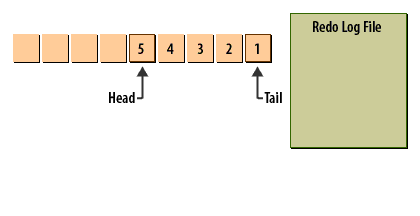
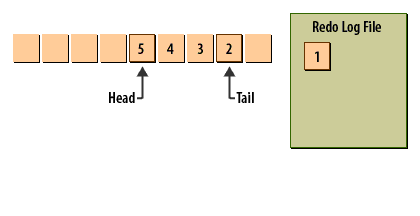
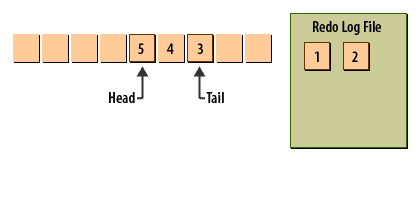
glibc-2.3.4-2.41

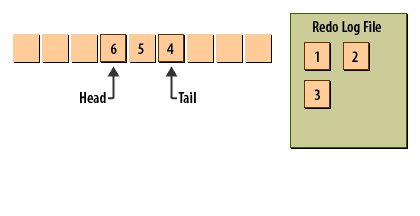
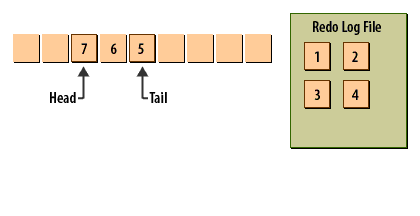
glibc-common-2.3.4-2.41

libaio-0.3.105-2

libgcc-3.4.6-10

The head command defaults to displaying the first 10 lines of the specified file. Like with tail, the default can be overridden and more lines or fewer lines can be displayed by adding a number as an option. For example, head -5 required\_packages.txt will display only the first five lines of the file.

1) As changes are made to the database, the head is advanced, and new entries are added to the redo log.  
2) As changes are made to the database, the head is advanced, and new entries are added to the redo log.  
3) As changes are made to the database, the head is advanced, and new entries are added to the redo log.  
4) As changes are made to the database, the head is advanced, and new entries are added to the redo log.  
5) At the same time, LGWR is always writing redo log records from the tail of the redo log.  
6) Here redo log record #1 has been written, and the tail pointer has been advanced.  
7) Now, redo log record #2 has been written, and the pointer advanced again.

8) These processes can occur simultaneously.  
9) The process continues as long as the instance is running.

## Specifying Redo Log Writing Process

Timely protection of application data requires use of the log writer process to propagate primary database modifications to one or more standby databases. This is achieved using the LGWR attribute of the LOG\_ARCHIVE\_DEST\_n initialization parameters.

Werbung

Werbung

|  |  |  |
| --- | --- | --- |
| Attribute | Example | Default |
| {LGWR|ARCH} | LOG\_ARCHIVE\_DEST\_3='SERVICE=stby1 LGWR' | ARCH |

### LGWR attribute

Choosing the LGWR attribute indicates that the log writer process (LGWR) will concurrently create the archived redo logs as the online redo log is populated. Depending on the configuration, this may require the log writer process to also transmit redo log files to remote archival destinations.

Choosing the ARCH attribute indicates that the (ARCn) archiver process will create archived redo logs on the primary database and also transmit redo logs for archival at specified destinations. This is the default setting. The LGWR and ARCH attributes are mutually exclusive. Therefore, you cannot specify the two attributes for the same destination.  
The LGWR attribute can be specified for individual destinations and this allows you to specify that the log writer process writes to redo logs and archives for some destinations while the archiver process archives redo logs to other destinations.

# Archiver (ARC0)

Werbung

## Storing and copying redo log files

The Achiver process makes a copy of each redo log file as it is filled, and stores that copy in an offline location. It is possible to run an Oracle database without archiving the redo logs, but archiving enables up to the minute recovery from a failure, so most production databases take advantage of it.  
The following Slide Show demonstrates what the Archiver process does:

**Archiver (ARC0)**

I examined my alert log and noticed that my archive process is failing with the *ARC0: Failed to archive log* message.  
**Question**: Can someone suggest how this *alert log error* can be corrected?

ARC0: Beginning to archive log# 2 seq# 313533

ARC0: Failed to archive log# 2 seq# 313533

Thur May 07 07:42:57 2015

ARC1: Failed to archive log# 3 seq# 313532

Thur May 07 07:42:57 2015

ARC1: Beginning to archive log# 2 seq# 313533

ARC1: Failed to archive log# 2 seq# 313533

Thur May 07 07:43:31 2015

ARC0: Beginning to archive log# 3 seq# 313532

ARC0: Failed to archive log# 3 seq# 313532

ARC0: Beginning to archive log# 2 seq# 313533

ARC0: Failed to archive log# 2 seq# 313533

**Answer**: This alert log message "failed to archive log" is not an "error", it is just an informational message noting that there was a delay when archiving your redo log, usually due to a disk I/O slowdown.  
You have not lost any *redo data*, but this error does indicate that you have a redo-related bottleneck, usually disk I/O.

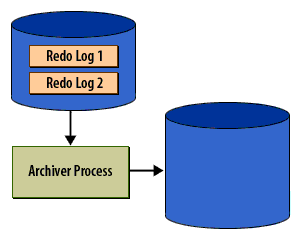
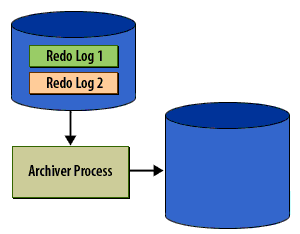
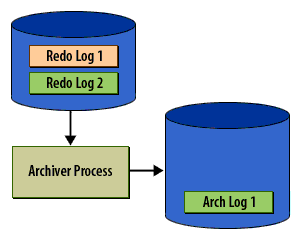
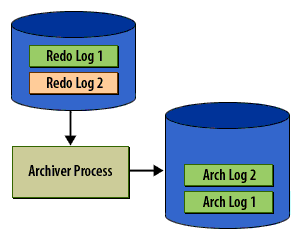
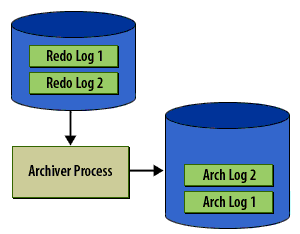
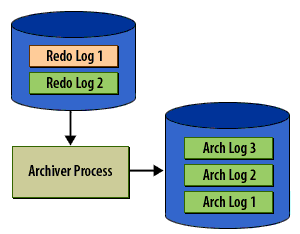
1. Make sure that your redo is on a separate disk spindle

Werbung

1. If necessary, move your redo to faster disk (SSD) devices.

It seems like your logfiles are not keeping up write activity and you should either

1. move your logfiles to speedy media to avoid write contention, or
2. try adding another log group members or resize redo log files with bigger number.

As changes are made to an Oracle database, a log of those changes is written to the redo log files.  
As each redo log is filled, the archiver process will begin to copy it to the archive.  
The archiver will more or less keep up with the log writer.  
If the log writer gets far enough ahead to fill up all the redo log files, then database users will be forced to wait while the archiver copies another file.  
When a database is running in archive log mode, then log writer will not overwrite a log file until it has been copied by the archiver.  
Once a log file has been copied, processing will resume. The archived log files preserve a record of all changes and is used for disaster recovery.

Like the Database Writer, you can have up to [**ten archiver processes**](https://www.relationaldbdesign.com/database-architecture/module4/multiple-archiver-processes.php) running for a single instance in Oracle.

Archiving is the operation of generating an archived redo log file. Archiving is either automatic or manual and is only possible when the database is in ARCHIVELOG mode. An archived redo log file includes the redo entries and the log sequence number of the identical member of the online redo log group. The archived redo log contains a copy of every group created since you enabled archiving.

## ARCHIVELOG mode

When Oracle is run in ARCHIVELOG mode, the ARC0-ARCt (Archiver) background processes make a copy of each redo log file before overwriting it. These archived redo log files are usually written to a disk device. The archived redo log files may also be written directly to a tape device, but this tends to be very operator-intensive. You can perform file system backups of a database while that database is open, provided the database is running in ARCHIVELOG mode. An online backup involves setting each tablespace (or the entire database) into a backup state, backing up the related datafiles, and then restoring the tablespaces to their normal state.

**ARCHIVE LOG LIST Command**

**Checking State of redo logs and archive logs**

There are times as a DBA, when you need to know the current state of the database relative to the redo logs and archive logs. You might want to find out something as simple as whether the database is in archivelog mode, or you may want to find out which redo log file is currently in use. Server Manager provides a command for this purpose. The command is ARCHIVE LOG LIST, and with it you can find out whether your database is in archive log mode, which redo log file is currently being written to, and the archive log destination. You run the ARCHIVE LOG LIST command from Server Manager. Here is an example:

SVRMGR> archive log list

Database log mode Archive Mode

Automatic archival Enabled

Archive destination /m01/oracle/oradata/prod/archive

Oldest online log sequence 20

Next log sequence to archive 21

Current log sequence 21

In this example, the first line tells you that the database is in archive log mode. The second line tells you that the Archiver process is running, and automatically copying redo log files when they are filled. The third line tells you the directory to which those log files are being copied. The last three lines tell you that the oldest log file still online is #20, and that the one currently being used is #21.

Oracle[**archivelog list command**](https://www.relationaldbdesign.com/database-architecture/module4/oracle-archivelog-list-command.php)

**ARCHIVE LOG LIST Command**

The SQL\*Plus command ARCHIVE LOG LIST displays archiving information for the connected instance. For example:

Werbung

SQL> ARCHIVE LOG LIST

Database log mode Archive Mode

Automatic archival Enabled

Archive destination D:\oracle\oradata\IDDB2\archive

Oldest online log sequence 11160

Next log sequence to archive 11163

Current log sequence 11163

This display tells you all the necessary information regarding the archived redo log settings for the current instance:

1. The database is currently operating in ARCHIVELOG mode.
2. Automatic archiving is enabled.
3. The archived redo log destination is D:\oracle\oradata\IDDB2\archive.
4. The oldest filled redo log group has a sequence number of 11160.
5. The next filled redo log group to archive has a sequence number of 11163.

Werbung

1. The current redo log file has a sequence number of 11163.

**ARCHIVE LOG**

**Syntax**: ARCHIVE LOG LIST  
Displays information about redo log files.

**LIST**

Requests a display that shows the range of redo log files to be archived, the current log file group's sequence number, and the current archive destination (specified by either the optional command text or by the initialization parameter LOG\_ARCHIVE\_DEST). If you are using both ARCHIVELOG mode and automatic archiving, the display might appear like:

ARCHIVE LOG LIST

Database log mode Archive Mode

Automatic archival Enabled

Archive destination /vobs/oracle/dbs/arch

Oldest online log sequence 221

Next log sequence to archive 222

Current log sequence 222

Since the log sequence number of the current log group and the next log group to archive are the same, automatic archival has archived all log groups up to the current one.  
If you are using ARCHIVELOG but have disabled automatic archiving, the last three lines might look like:

Oldest online log sequence 222

Next log sequence to archive 222

Current log sequence 225

If you are using NOARCHIVELOG mode, the "next log sequence to archive" line is suppressed. The log sequence increments every time the Log Writer begins to write to another redo log file group; it does not indicate the number of logs being used. Every time an online redo log file group is reused, the contents are assigned a new log sequence number. **Usage:** You must be connected to an open Oracle database as SYSOPER, or SYSDBA

## Checkpoint process (CKPT)

The Checkpoint process (not surprisingly) controls database checkpoints. Every 3 seconds the Checkpoint process determines the earliest redo log entry for which changes have not been written to the database. This becomes the checkpoint, and it is recorded in the control file and in all the datafiles. The following Slide Show illustrates this process:

## Checkpoint process (CKPT)

You can see the Oracle background processes with this query:

select \*

from

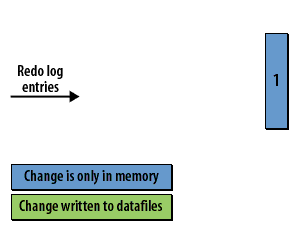
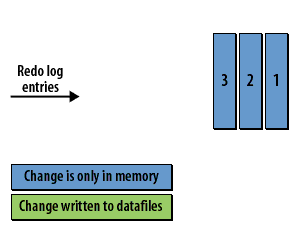
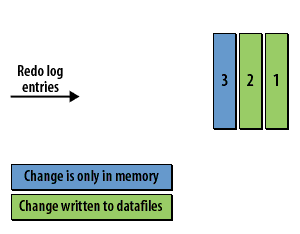
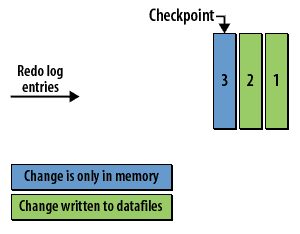
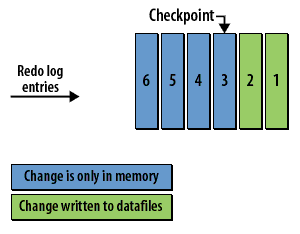
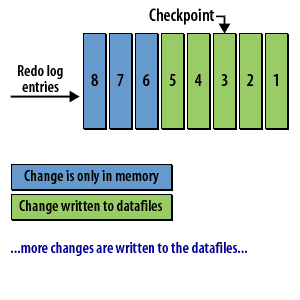
v$session

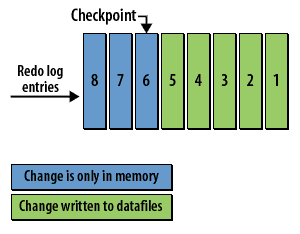
where

type ='BACKGROUND';

Here are some of the most important Oracle background processes:

|  |  |
| --- | --- |
| ARCH | (Optional) Archive process writes filled redo logs to the archive log location(s). In RAC, the various ARCH processes can be utilized to ensure that copies of the archived redo logs for each instance are available to the other instances in the RAC setup should they be needed for recovery. |
| CJQ - Job Queue Process | (CJQ) - Used for the job scheduler. The job scheduler includes a main program (the coordinator) and slave programs that the coordinator executes. The parameter job\_queue\_processes controls how many parallel job scheduler jobs can be executed at one time. |
| CKPT: | Checkpoint process writes checkpoint information to control files and data file headers |
| CQJ0: | Job queue controller process wakes up periodically and checks the job log. If a job is due, it spawns Jnnnn processes to handle jobs. |
| DBWR: | Database Writer or Dirty Buffer Writer process is responsible for writing dirty buffers from the database block cache to the database data files. Generally, DBWR only writes blocks back to the data files on commit, or when the cache is full and space has to be made for more blocks. The possible multiple DBWR processes in RAC must be coordinated through the locking and global cache processes to ensure efficient processing is accomplished. |
| FMON : | The database communicates with the mapping libraries provided by storage vendors through an external non-Oracle Database process that is spawned by a background process called FMON. FMON is responsible for managing the mapping information. When you specify the FILE\_MAPPING initialization parameter for mapping data files to physical devices on a storage subsystem, then the FMON process is spawned. |
| LGWR: | Log Writer process is responsible for writing the log buffers out to the redo logs. In RAC, each RAC instance has its own LGWR process that maintains that instance thread of redo logs. |
| LMON: | Lock Manager process |
| MMON: | The Oracle 10g background process to collect statistics for the Automatic Workload Repository (AWR). |
| MMNL: | This process performs frequent and lightweight manageability-related tasks, such as session history capture and metrics computation. |
| MMAN: | is used for internal database tasks that manage the automatic shared memory. MMAN serves as the SGA Memory Broker and coordinates the sizing of the memory components. |
| PMON : | Process Monitor process recovers failed process resources. If MTS (also called Shared Server Architecture) is being utilized, PMON monitors and restarts any failed dispatcher or server processes. In RAC, PMON role as service registration agent is particularly important. |
| Pnnn : | (Optional) Parallel Query Slaves are started and stopped as needed to participate in parallel query operations. |
| RBAL: | This process coordinates rebalance activity for disk groups in an Automatic Storage Management instance. |
| SMON: | System Monitor process recovers after instance failure and monitors temporary segments and extents. SMON in a non-failed instance can also perform failed instance recovery for other failed RAC instance. |
| WMON: | The "wakeup" monitor process |

As changes are made to the database, they are quickly recorded in the redo log, but are not immediately written out to the datafiles.  
We have three redo log entries. They are all shown in blue, because DBWR has not yet written any of the changes to the datafiles.  
The database writer will write out some changes. Here, the changes for entires 1 and 2 have been written to the datafiles.  
A checkpoint is recorded every three seconds. Here the checkpoint is redo log entry 3, because all prior changes have been written.  
This process continues. More redo records are written.  
More changes are written to the datafiles.

The checkpoint is advanced.

**The checkpoint and data recovery**

Recording the checkpoint is important for recovery purposes. The checkpoint identifies the first redo log entry that Oracle will need to process in a recovery situation. All changes related to previous redo log entries have been written to the disk.

**Checkpoint Process (CKPT)**

When a checkpoint occurs, Oracle must update the headers of all datafiles to record the details of the checkpoint. This is done by the CKPT process.

Werbung

The CKPT process does not write blocks to disk; DBWn always performs that work. The statistic DBWR checkpoints displayed by the System\_Statistics monitor in Enterprise Manager indicates the number of checkpoint requests completed.

**Considerations with Bigfile Tablespaces**

1. Bigfile tablespaces are intended to be used with Automatic Storage Management or other logical volume managers that support dynamically extensible logical volumes and striping or RAID.
2. Avoid creating bigfile tablespaces on a system that does not support striping because of negative implications for parallel execution and RMAN backup parallelization.
3. Avoid using bigfile tablespaces if there could possibly be no free space available on a disk group, and the only way to extend a tablespace is to add a new datafile on a different disk group.
4. Using bigfile tablespaces on platforms that do not support large file sizes is not recommended and can limit tablespace capacity. Refer to your operating system specific documentation for information about maximum supported file sizes.
5. Performance of database opens, checkpoints, and DBWR processes should improve if data is stored in bigfile tablespaces instead of traditional tablespaces. However, increasing the datafile size might increase time to restore a corrupted file or create a new datafile.

**Oracle Processes such as Database Writer, Log Writer, System Monitor, Process Monitor, Archiver**

**Main processes**

Identify other Oracle processes that may be running on your server. The processes that you have learned about so far in this module are the major ones that you need to be concerned about when managing an Oracle instance. These include:

1. The Database Writer
2. The Log Writer
3. The System Monitor
4. The Process Monitor
5. The Archiver

**Other processes**

There are other processes however, that you may see running, depending on how you have configured your database. Here are some of them:

|  |  |
| --- | --- |
| LCKO:Lock processes: | Used with Oracle Parallel Server to provide inter-instance locking. This allows two instances operating on the same database to coordinate their activities. |
| SNPO: Job Queue processes: | Job Queue processes are used to run scheduled jobs. The replication option uses these to refresh snapshots, hence the SNP abbreviation. SNP processes also start jobs scheduled with the DBMS\_JOBS package. |
| QMNO:Queue Monitor processes: | Queue Monitor processes monitor the message queues as part of Oracle's Advanced Queuing feature. |
| Dnnn-Dispatcher processes: | Dispatcher processes exist when the multi-threaded server option is being used. Dispatchers link up incoming connections with available shared server processes. |
| Snnn-Shared Server processes: | Shared Server processes are also used with the multi-threaded server option. Multiple user connections share a single server process,k reducing the demand on server resources. |

**Latches**

A latch is a simple, low-level serialization mechanism that coordinates multiuser access to shared data structures, objects, and files. Latches protect shared memory resources from corruption when accessed by multiple processes. Specifically, latches protect data structures from the following situations:

1. Concurrent modification by multiple sessions
2. Being read by one session while being modified by another session
3. Deallocation (aging out) of memory while being accessed

Typically, a single latch protects multiple objects in the SGA. For example, background processes such as DBW and LGWR allocate memory from the shared pool to create data structures. To allocate this memory, these processes use a shared pool latch that serializes access to prevent two processes from trying to inspect or modify the shared pool simultaneously. After the memory is allocated, other processes may need to access shared pool areas such as the library cache, which is required for parsing. In this case, processes latch only the library cache, not the entire shared pool. Unlike enqueue latches such as row locks, latches do not permit sessions to queue. When a latch becomes available, the first session to request the latch obtains exclusive access to it. The phenomenon of latch spinning occurs when a process repeatedly requests a latch in a loop, whereas latch sleeping occurs when a process releases the CPU before renewing the latch request. Typically, an Oracle process acquires a latch for an extremely short time while manipulating or looking at a data structure. For example, while processing a salary update of a single employee, the database may obtain and release thousands of latches. The implementation of latches is operating system-dependent, especially in respect to whether and how long a process waits for a latch. An increase in latching means a decrease in concurrency. For example, excessive hard parse operations create contention for the library cache latch. The V$LATCH view contains detailed latch usage statistics for each latch, including the number of times each latch was requested and waited for.

**Oracle Instance Architecture Conclusion**

This is the end of the module on Oracle processes. This module discussed the purpose of the Oracle significant background processes.  
You should understand the difference between a database and an instance, and you should also be able to do the following:

1. Explain the difference between a database and an instance
2. Identify the major background processes of an Oracle instance, and explain their purpose
3. Display a list of the background processes that are currently running
4. Identify the instances running on your system based on a listing of process names

**Glossary**

This module introduced you to the following terms:

1. *asynchronously:*When applied to processes, Oracle processes in particular, to execute asynchronously means to execute at the same time as other processes, without having to take turns
2. *buffer I/O:*A logical I/O, also known as a buffer I/O, refers to reads and writes of buffers in the buffer cache.
3. *crash recovery:*The process of redoing changes that were lost during a system crash because the affected data blocks were contained only in memory. During crash recovery, the redo log files are read, and the changes reapplied to the datafiles

Werbung

1. *dirty read:* A transaction reads data that has been written by another transaction that has not been committed yet.
2. *grepping:*The process, used under UNIX, of filtering the output of one command through the grep command in order to restrict output to specific items of interest.
3. *head of the log*A pointer to the oldest redo log entry, in the redo log buffer, that has not yet been written to the redo log files. The log writer process always writes from the head-end of the log.
4. *Least Recently Used List:*A list of pointers to database buffers that is maintained by Oracle in the SGA. The pointers in the LRU list are arranged in order based on the time interval since a buffer was last accessed.
5. *LRU List:*Least Recently Used List. A list of pointers to database buffers that is maintained by Oracle in the SGA. The pointers in the LRU list are arranged in order based on the time interval since a buffer was last accessed.
6. *(MTS)multi-threaded server:* The (MTS) multi-threaded server has been replaced by the Oracle *shared server* since Oracle 9i.
7. *recovery:*The process of reading entries from the redo log files (including the archived redo log files) and using that information to replay changes that have been made to the database. Recovery is often used to bring a database up-to-date after it has been restored from a backup.
8. *redo log entry:*chronological records of database actions used during database recoveries.
9. *redo log buffer*
10. *service*
11. *SGA*
12. *SID:*The system identifier (SID) is a unique name for an Oracle database instance on a specific host.
13. *tail of the log*
14. *temporary segments*
15. *thread:*A thread is an independent execution path within a program.

**Connecting to the Database with SQL\*Plus**

Oracle Database includes the following components:

1. The Oracle Database instance, which is a collection of processes and memory
2. A set of disk files that contain user data and system data

When you connect with SQL\*Plus, you are connecting to the Oracle instance. Each instance has an instance ID, also known as a system ID (SID). Because there can be more than one Oracle instance on a host computer, each with its own set of data files, you must identify the instance to which you want to connect. For a local connection, you identify the instance by setting operating system environment variables. For a remote connection, you identify the instance by specifying a network address and a database service name. For both local and remote connections, you must set environment variables to help the operating system find the SQL\*Plus executable and to provide the executable with a path to its support files and scripts.

To connect to an Oracle instance with SQL\*Plus, therefore, you must complete the following steps:

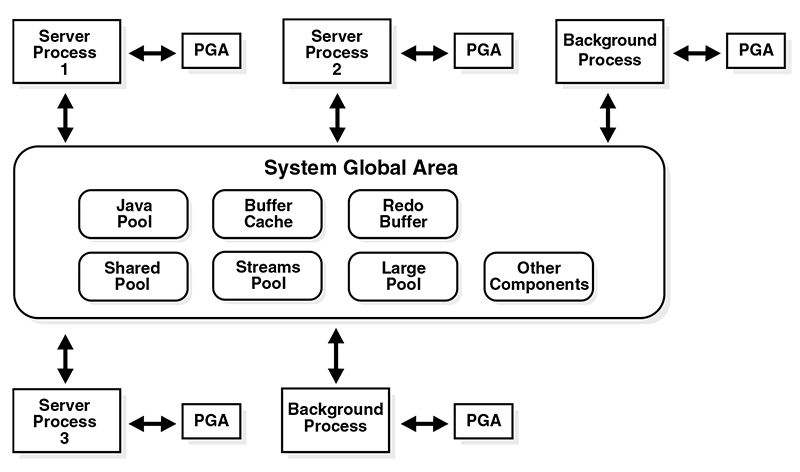
1. Step 1: Open a Command Window
2. Step 2: Set Operating System Environment Variables
3. Step 3: Start SQL\*Plus
4. Step 4: Submit the SQL\*Plus CONNECT Statement

**Understanding Oracle's Memory Architecture**

Now that you know about files and processes, it is time to learn about the last piece of the architecture puzzle, which is memory.  
Oracle uses several memory structures: the

1. System Global Area,
2. Program Gobal Areas,
3. and Sort Areas.

See Figure 1 below

Figure 1: Oracle Database Memory Structures consisting of 1) Server Processes and 2) Background Process interacting with the System Global Area

Understanding and properly sizing the System Global Area, Program Global Areas, and Sort Areas can have big impact on overall database performance.  
The System Global Area is the most significant structure, and the most important to understand. You've already seen, in the previous module, that it's divided into several smaller areas known as the database buffer cache, the redo log buffer, and the shared pool. This module goes into more detail about each of those structures, particularly the shared pool.

Werbung

When you are done with this module, you should be able to:

1. Describe the memory structures that make up the SGA
2. Describe how the database buffer cache manages data blocks
3. Describe the purpose of the three types of buffer pools
4. Describe the function of the redo log buffer.
5. Identify the function of the large pool
6. Describe how each session gets its own memory
7. List the contents of the PGA
8. Describe how Oracle uses memory for sorting

The knowledge that you gain in this lesson will enable you to understand many of the tuning issues that will arise when you manage an Oracle database.

**Alternate meaning for SGA**

**Shared Global Area is same as System Global Area**

The acronym SGA stands for System Global Area. Because it's a shared memory structure, and because the word “shared” also begins with an “S,” people sometimes refer to the SGA as the Shared Global Area. This is such a common occurrence that Oracle even makes mention of it in their manuals. I always use the correct term, but not everyone will. So if you hear someone talk about the Shared Global Area, just mentally translate that to System Global Area.

**Automatic Memory Management**

Oracle strongly recommends the use of automatic memory management to manage the memory on your system. Automatic memory management enables Oracle Database to automatically manage and tune the instance memory. Automatic memory management can be configured using a target memory size initialization parameter (MEMORY\_TARGET) and a maximum memory size initialization parameter (MEMORY\_MAX\_TARGET). The database tunes to the target memory size, redistributing memory as needed between the system global area (SGA) and the instance program global area (instance PGA). Before setting any memory pool sizes, consider using the automatic memory management feature of Oracle Database. If you must configure memory allocations, consider using the Memory Advisor for managing memory

**Considering Multiple Buffer Pools**

A single default buffer pool is generally adequate for most systems. However, users with detailed knowledge of an application's buffer pool might benefit from configuring multiple buffer pools. With segments that have atypical access patterns, store blocks from those segments in two different buffer pools: the KEEP pool and the RECYCLE pool. A segment's access pattern may be atypical if it is constantly accessed (that is, hot) or infrequently accessed (for example, a large segment accessed by a batch job only once a day). Multiple buffer pools let you address these differences. You can use a KEEP buffer pool to maintain frequently accessed segments in the buffer cache, and a RECYCLE buffer pool to prevent objects from consuming unnecessary space in the cache. When an object is associated with a cache, all blocks from that object are placed in that cache. Oracle Database maintains a DEFAULT buffer pool for objects that have not been assigned to a specific buffer pool. The default buffer pool is of size DB\_CACHE\_SIZE. Each buffer pool uses the same Least Recently Used (LRU) replacement policy (for example, if the KEEP pool is not large enough to store all of the segments allocated to it, then the oldest blocks age out of the cache).  
By allocating objects to appropriate buffer pools, you can:

1. Reduce or eliminate I/Os
2. Isolate or limit an object to a separate cache

**Databases and Instances**

Many Oracle practitioners use the terms instance and database interchangeably. In fact, an instance and a database are different entities, but they are still related. This distinction is important because it provides insight into *Oracle's architecture*. In Oracle, the term database refers to the physical storage of information, and the term instance refers to the software executing on the server that provides access to the information in the database. The instance runs on the computer or server; the database is stored on the disks attached to the server.

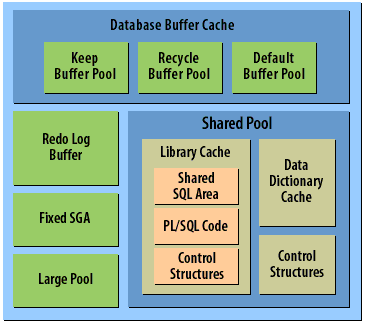
1. The database is physical: it consists of files stored on disks.
2. The instance is logical: it consists of in-memory structures and processes on the server.

For example, Oracle uses an area of shared memory called the System Global Area (SGA) and a private memory area for each process called the Program Global Area (PGA). An instance can be part of one and only one database, although multiple instances can be part of the same database. Instances are temporal, but databases, with proper maintenance, last forever. Users do not directly access the information in an Oracle database. Instead, they pass requests for information to an Oracle instance.

The System Global Area, or SGA, is the heart of any Oracle instance. All the various Oracle processes communicate with the SGA in one way or another.

**Memory Structures in SGA**

The previous module introduced you to some of the major structures in the SGA. This module will go a bit deeper and discuss the topics in more detail. Here is a more detailed version of a MouseOver you saw earlier.



1. Database Buffer Cache: Holds data blocks read from disk
2. Keep Buffer Pool: Data blocks that are kept in memory
3. Recycle Buffer Pool: Data blocks that are recycled as soon as possible
4. Default Buffer Pool: Data blocks managed in the default fashion
5. Redo Log Buffer: Holds redo log entries waiting to be written to disk
6. Fixed SGA: Contains general information about the state of the database and the instance
7. Large Pool: An optional memory area used for backup and restore operations
8. Shared Pool: Contains memory structures related to SQL execution
9. Library Cache: Contains the shared SQL area and PL/SQL code
10. Data Dictionary Cache: An area in the Shared Pool where Oracle stores frequently accessed data dictionary information so that it does not need to be continuously reread from disk.
11. Control Structures: Library cache, Locks, cache handles, etc.
12. Control Structures: Shared pool, character set conversion memory, etc.
13. Shared SQL Area: SQL statements and execution plans
14. PL/SQL code, packages, procedures, and functions
15. Database Buffer Cache: Database Buffer Cache: Holds data blocks read from disk and
16. Shared Pool:Contains memory structures related to SQL execution

**Database Buffer Cache**

The database buffer cache, also called the buffer cache, is the memory area that stores copies of data blocks read from data files. A buffer is a main memory address in which the buffer manager temporarily caches a currently or recently used data block. All users concurrently connected to a database instance share access to the buffer cache.

**Purpose of the Database Buffer Cache**

Oracle Database uses the buffer cache to achieve the following goals:

1. Optimize physical I/O: The database updates data blocks in the cache and stores metadata about the changes in the redo log buffer. After a COMMIT, the database writes the redo buffers to the online redo log but does not immediately write data blocks to the data files. Instead, database writer (DBW) performs lazy writes in the background.
2. Keep frequently accessed blocks in the buffer cache and write infrequently accessed blocks to disk  
   When Database Smart Flash Cache (flash cache) is enabled, part of the buffer cache can reside in the flash cache. This buffer cache extension is stored on one or more flash disk devices, which are solid state storage devices that uses flash memory. The database can improve performance by caching buffers in flash memory instead of reading from magnetic disk. Use the DB\_FLASH\_CACHE\_FILE and DB\_FLASH\_CACHE\_SIZE initialization parameters to configure multiple flash devices. The buffer cache tracks each device and distributes buffers to the devices uniformly.

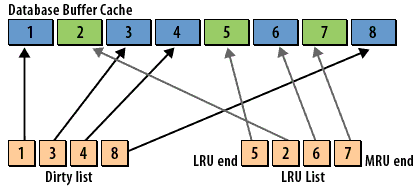
**Oracle Database Buffer Cache**

**The database buffer cache holds data blocks**

The *database buffer cache* is usually the largest structure within the SGA. Its purpose is to hold as many data blocks in memory as possible in order to minimize the number of reads that an instance needs to perform. The larger the buffer cache, the better the chance that any given data block will already be in memory when it is needed.

**Dirty list and LRU list**

The buffer size always matches the database block size, which is specified in the database's initialization file. The buffers are organized into two lists: the *dirty list*, and the *LRU List*. LRU is an acronym that stands for Least Recently Used. Here's a diagram that shows how all this might look after an instance has been running for awhile. Move your mouse over the diagram to learn how the buffers are being used.



1. Blue buffers 1,3,4,8: Dirty buffer
2. Blue buffers 1,3,4,8: Dirty buffer
3. Blue buffer 6: Will be moved to the dirty list
4. Green buffer 5: The least recently used buffer
5. Green buffer 7: The most recently used buffer
6. Green buffer 2: The third most recently used buffer
7. Dirty List 1,3,4,8: Pointers to the dirty buffers
8. LRU List 5: Pointer to the least recently used buffer

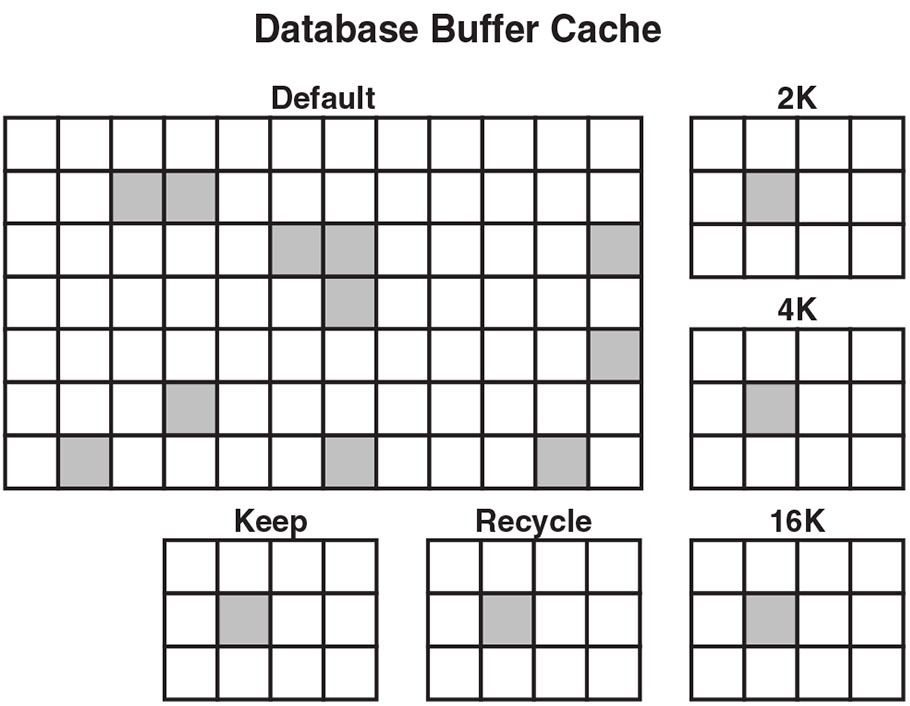
Werbung

1. LRU List 7: Pointer to the most recently used buffer
2. LRU List 2: Pointer to the third most recently used buffer
3. LRU List 6: Pointer to the second most recently used buffer
4. Blue buffers 1,3,4,8: Dirty buffer
5. Blue buffers 1,3,4,8: Dirty buffer
6. Blue buffers 1,3,4,8: Dirty buffer

[**Database Buffer Cache**](https://www.relationaldbdesign.com/database-architecture/module5/dataBaseBuffer-cache.php)

The dirty list points to all the buffers that have been modified and that need to be written back to disk. Sometimes dirty buffers will be found in the LRU List as well, but eventually they too will make it to the dirty list. All the other buffers are contained in the LRU list.  
The LRU list has two ends: the least recently used end and the most recently used end. Every time a buffer is written to or read from, it is moved to the most recently used end of the LRU list. Buffers on the least end of the LRU list are overwritten when new data needs to be read from disk.

A database has a standard block size. You can create a tablespace with a block size that differs from the standard size. Each non-default block size has its own pool. Oracle Database manages the blocks in these pools in the same way as in the default pool. Figure 5-3 shows the structure of the buffer cache when multiple pools are used. The cache contains default, keep, and recycle pools. The default block size is 8 KB. The cache contains separate pools for tablespaces that use the nonstandard block sizes of 2 KB, 4 KB, and 16 KB.

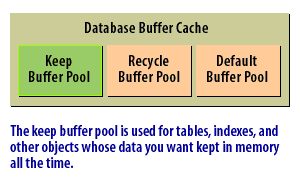
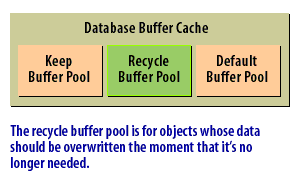
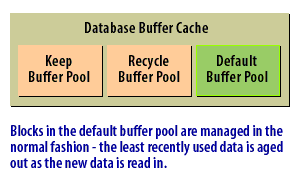
Figure 5-3: Database Buffer Cache

**Buffer cache affects performance**

You can affect the performance of a database instance by changing the size of the buffer cache. Make it too small, and performance will suffer because of excessive disk I/O. Make it too large, and you will be wasting memory that could be more profitably used.  
The ideal is to increase the buffer cache size until you find the point where further increases do not result in any further improvements.

**Three Types of Buffer Pools**

Werbung

1) The **keep buffer pool** is used for tables, indexes, and other objects whose data you want kept in memory all the time.  
2) The **recycle buffer pool** is for objects whose data should be overwritten the moment that it's no longer needed.  
3) Blocks in the **default buffer pool** are managed in the normal fashion - the least recently used data is aged out as the new data is read in.

**Other pools in the SGA**

The SGA includes several other pools:

1. *Large pool:* Provides memory allocation for various I/O server processes, backup, and recovery, and provides session memory where shared servers and Oracle XA for transaction processing are used.
2. *Java pool:* Provides memory allocation for Java objects and Java execution, including data in the Java Virtual Machine in the database.
3. *Streams pool:* Provides memory allocation used to buffer Oracle Streams queued messages in the SGA instead of in database tables and provides memory for capture and apply.

Dynamic initialization parameters available for these pools include LARGE\_POOL\_SIZE, JAVA\_POOL\_SIZE, and STREAMS\_POOL\_SIZE.

These are automatically set if MEMORY\_TARGET or SGA\_TARGET is specified.

**Three types of Buffer Pools**

Prior to the release of Oracle, the database buffer cache consisted of one large pool of buffers that were all treated the same way. Oracle introduced the capability of having multiple buffer pools for different purposes. Three types of buffer pools are supported:

1. *Keep buffer pool*
2. *Recycle buffer pool*
3. *Default buffer pool*

The following Slideshow explains the purpose of each:

**Assigning objects to Pools**

When you create a database object, such as a table, you have the option of assigning it to one of these buffer pools. Here are some guidelines for assigning objects to pools:

|  |  |
| --- | --- |
| Keep buffer pool | Assign small code tables that are referenced frequently to this pool. |
| Recycle buffer pool: | Assign large objects that are read randomly, and that you don't want cluttering the cache to this pool. |
| Default buffer pool: | All other objects, including those that you don't explicitly assign, should go to the default buffer pool. |

Do not worry too much about the specifics of placing a given object in one of these pools. For now, it's enough to understand that they exist, and for you to have some idea of their purpose. Later, when you find yourself creating tables and indexes in a real database, keep these buffer pools in mind. If you have frequently accessed objects that fall into one of the categories listed above, you may be able to improve performance by assigning that object to the appropriate buffer pool.

**Buffer Pools**

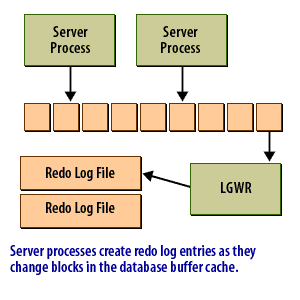
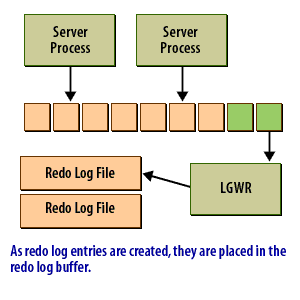
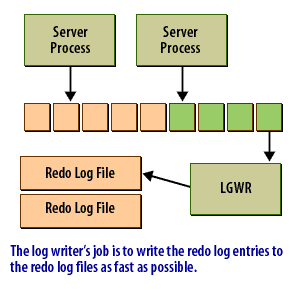
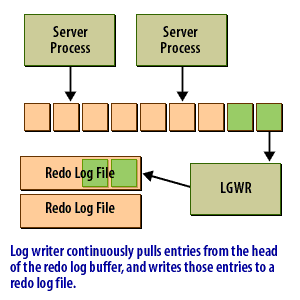
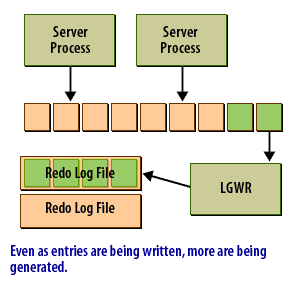
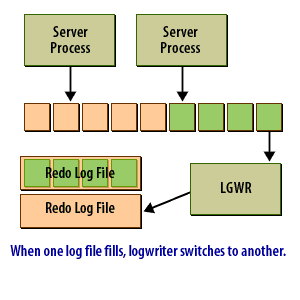
A buffer pool is a collection of buffers. The database buffer cache is divided into one or more buffer pools. You can manually configure separate buffer pools that either keep data in the buffer cache or make the buffers available for new data immediately after using the data blocks. You can then assign specific schema objects to the appropriate buffer pool to control how blocks age out of the cache. The possible buffer pools are as follows:

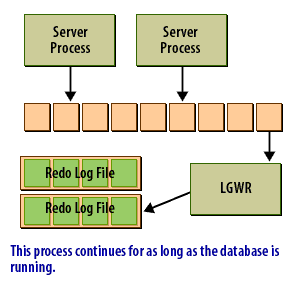
1. Keep pool: This pool is intended for blocks that were accessed frequently, but which aged out of the default pool because of lack of space. The goal of the keep buffer pool is to retain objects in memory, thus avoiding I/O operations.
2. Recycle pool: This pool is intended for blocks that are used infrequently. A recycle pool prevent objects from consuming unnecessary space in the cache.
3. Default pool: This pool is the location where blocks are normally cached. Unless you manually configure separate pools, the default pool is the only buffer pool.

# Function of Oracle Redo Log Buffer

The redo log buffer is an area of memory set aside to hold redo log entries long enough for the Log Writer process to write them to the redo log files. Ideally, the log writer would write the redo log as fast as new entries were generated. Then you would not need a buffer. In reality, that's simply not feasible. This SlideShow shows conceptually how the redo log buffer is used:

# Oracle Redo Log Buffer

1) Server processes create redo log entries as they change blocks in database buffer cache.  
2) As redo log entries are created, they are placed in the redo log buffer  
3) The log writer's job is to write the redo log entries to the redo log files as fast as possible.  
4) Log writer continuously pulls entries from the head of the redo log buffer, and writes those entries to a redo log file.  
5) Even as entries are being written, more are being generated  
6) When one log file fills, logwriter switches to another



7) This process continues for as long as the database is running

## Redo log buffer

The redo log buffer caches redo information until it is written to the physical redo log files stored on a disk. This buffer also improves performance. Oracle caches the redo until it can be written to a disk at a more optimal time, which avoids the overhead of constantly writing the redo logs to disk.

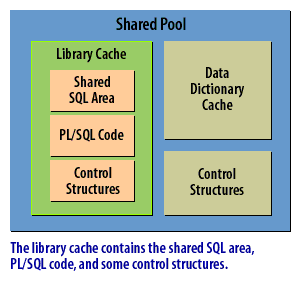
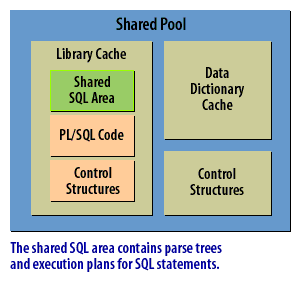
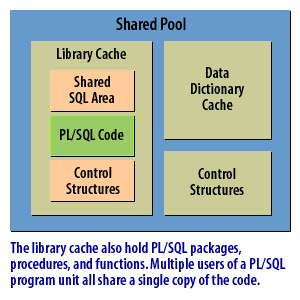
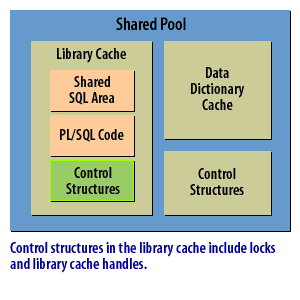
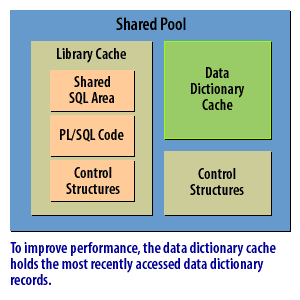
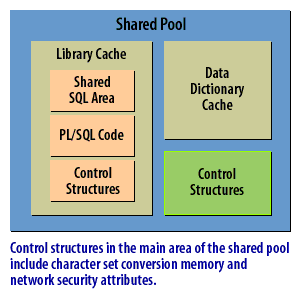
The redo log buffer holds the most recent changes to the data blocks in the datafiles. When the redo log buffer is one-third full, or every three seconds, Oracle writes redo log records to the redo log files. As of Oracle 10g, the LGWR process will write the redo log records to the redo log files when 1MB of redo is stored in the redo log buffer. The entries in the redo log buffer, once written to the redo log files, are critical to database recovery if the instance crashes before the changed data blocks are written from the buffer cache to the datafiles. A user's committed transaction is not considered complete until the redo log entries have been successfully written to the redo log files.

## What comprises the shared pool?

The shared pool is an area of shared memory that contains information related to the execution of SQL statements and PL/SQL code. This slide show will tell you a little about the three major structures that make up the shared pool:

# Oracle Shared pool

Werbung

1) The library cache contains the shared SQL area, PL/SQL code, and some control structures.  
2) The shared SQL area contains parse trees, and execution plans for SQL statements.  
3) The library cache holds PL/SQL packages, procedures, and functions. Multiple users of a PL/SQL program unit all share a single copy of the code  
4) Control structures in the library cache include locks and library cache handles  
5) To improve performance, the data dictionary cache holds the most recently accessed data dictionary records  
6) Control structures in the main area of the shared pool include character set conversion memory and network security attributes

## Shared Pool

The shared pool contains two major subcaches: the 1) library cache and 2) data dictionary cache. The shared pool is sized by the SHARED\_POOL\_SIZE initialization parameter. This is another dynamic parameter that can be resized as long as the total SGA size is less than SGA\_MAX\_SIZE or SGA\_TARGET.

### Library Cache

The library cache holds information about SQL and PL/SQL statements that are run against the database. In the library cache, because it is shared by all users, many different database users can potentially share the same SQL statement. Along with the SQL statement itself, the execution plan and parse tree of the SQL statement are stored in the library cache. The second time an identical SQL statement is run, by the same user or a different user, the execution plan and parse tree are already computed, improving the execution time of the query or DML statement. If the library cache is sized too small, the execution plans and parse trees are flushed out of the cache, requiring frequent reloads of SQL statements into the library cache. See Chapter 8 for ways to monitor the efficiency of the library cache.

### Data Dictionary Cache

The data dictionary is a collection of database tables, owned by the SYS and SYSTEM schemas, that contain the metadata about the database, its structures, and the privileges and roles of database users. The data dictionary cache holds a subset of the columns from data dictionary tables after first being read into the buffer cache. Data blocks from tables in the data dictionary are used continually to assist in processing user queries and other DML commands. If the data dictionary cache is too small, requests for information from the data dictionary will cause extra I/O to occur; these I/O-bound data dictionary requests are called recursive calls and should be avoided by sizing the data dictionary cache correctly.

### Parsing and execution plans

The shared SQL area is a particularly important part of the shared pool. When you send a SQL statement to the database, Oracle must determine how to execute that statement. Oracle needs to determine what tables are involved, which indexes to use, if any, what the join order should be, and so forth. This process is called parsing, and typically involves a number of recursive queries to the data dictionary.

**Characters must match in Shared Pool**

Oracle is very picky when it comes to deciding whether a newly submitted SQL statement matches one that is already in the shared SQL area.  
Character case, white space, and comments are all significant. Oracle matches two statements based on a character-by-character comparison, not by comparing the semantics, or meaning, of the statements.

The following SQL statements, which are all semantically the same, would not be considered identical:

1. SELECT SYSDATE FROM DUAL
2. select sysdate from dual
3. select sysdate from dual;
4. select sysdate /\* this is a comment \*/ from dual

The shared pool caches various types of program data. For example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information. The shared pool is involved in almost every operation that occurs in the database. For example, if a user executes a SQL statement, then Oracle Database accesses the shared pool. The shared pool is divided into several subcomponents.

**Shared SQL Areas**

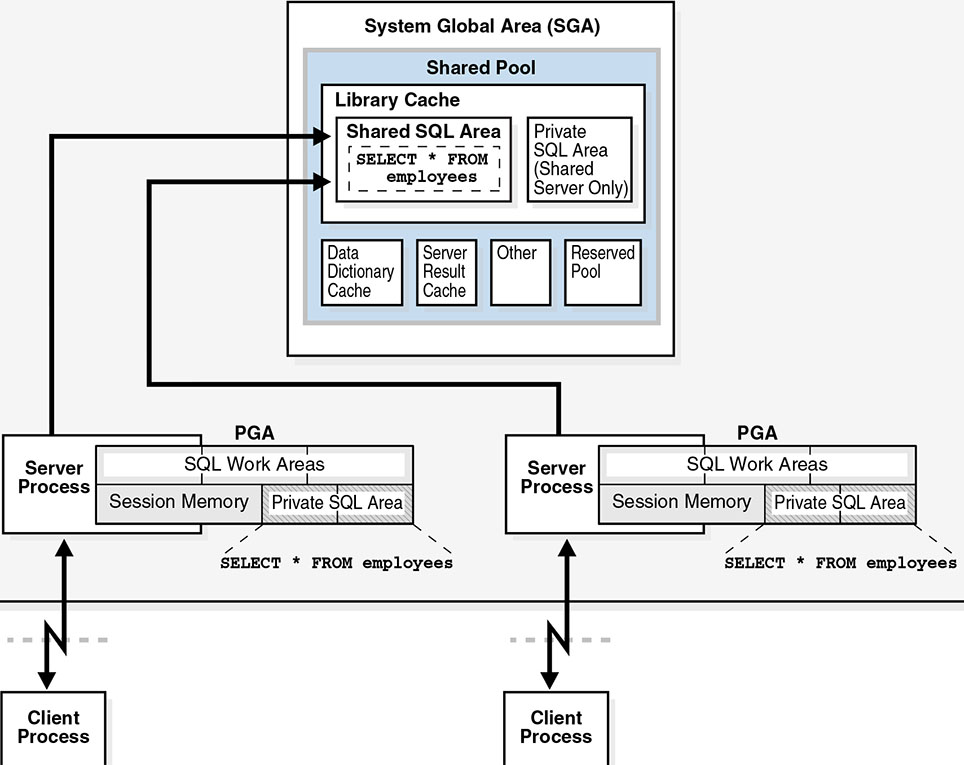
The database represents each SQL statement that it runs in the following SQL areas:

1. **Shared SQL area:** The database uses the shared SQL area to process the first occurrence of a SQL statement. This area is accessible to all users and contains the statement parse tree and execution plan. Only one shared SQL area exists for a unique statement .
2. **Private SQL area:** Each session issuing a SQL statement has a private SQL area in its PGA (see "Private SQL Area" on page 14-5). Each user that submits the same statement has a private SQL area pointing to the same shared SQL area. Thus, many private SQL areas in separate PGAs can be associated with the same shared SQL area.

The database automatically determines when applications submit similar SQL statements. The database considers both SQL statements issued directly by users and applications and recursive SQL statements issued internally by other statements. The database performs the following steps:

1. Checks the shared pool to see if a shared SQL area exists for a syntactically and semantically identical statement:  
   a) If an identical statement exists, then the database uses the shared SQL area for the execution of the subsequent new instances of the statement, thereby reducing memory consumption.  
   b) If an identical statement does not exist, then the database allocates a new shared SQL area in the shared pool. A statement with the same syntax but different semantics uses a child cursor. In either case, the private SQL area for the user points to the shared SQL area that contains the statement and execution plan.
2. Allocates a private SQL area on behalf of the session: The location of the private SQL area depends on the connection established for the session. If a session is connected through a shared server, then part of the private SQL area is kept in the SGA.

Figure 5-7 shows a dedicated server architecture in which two sessions keep a copy of the same SQL statement in their own PGAs. In a shared server, this copy is in the UGA, which is in the large pool or in the shared pool when no large pool exists.

Figure 5-7:Private SQL Areas and Shared SQL Area

**Referenced objects must match**

In addition to case, white space, and comments, Oracle also checks to be sure that the objects referenced by the two statements are the same.  
Consider the case where users Jenny and Jeff each own identical tables named coin. Jenny and Jeff each issue the following SQL statement:

select \* from coin

In Jenny's case, the object coin refers to the table jenny.coin. In Jeff's case, it refers to the table jeff.coin. Even though the two statements match when compared on a character basis, the objects referred to are different. Because the objects are different, Oracle will not consider the statements to be identical.

**Identify Function of Large Pool**

**Large memory allocations**

The large pool is an optional memory area. It provides an area of memory from which large allocations can be made. Oracle's backup and restore utilities typically allocate buffers that are hundreds of kilobytes in size. These will be allocated in the large pool if one is present. The *multi-threaded server* will also take advantage of the large pool, allocating session memory there instead of in the shared pool, thus leaving more of the shared pool open for SQL statements and execution plans.

**Configuring Large Pool**

A large pool will only be present if the DBA has used the LARGE\_POOL\_SIZE initialization parameter to configure one. For example, to allocate a large pool of 10 megabytes, you would add the following line to your database's initialization file:

LARGE\_POOL\_SIZE = 10M

**Large Pool**

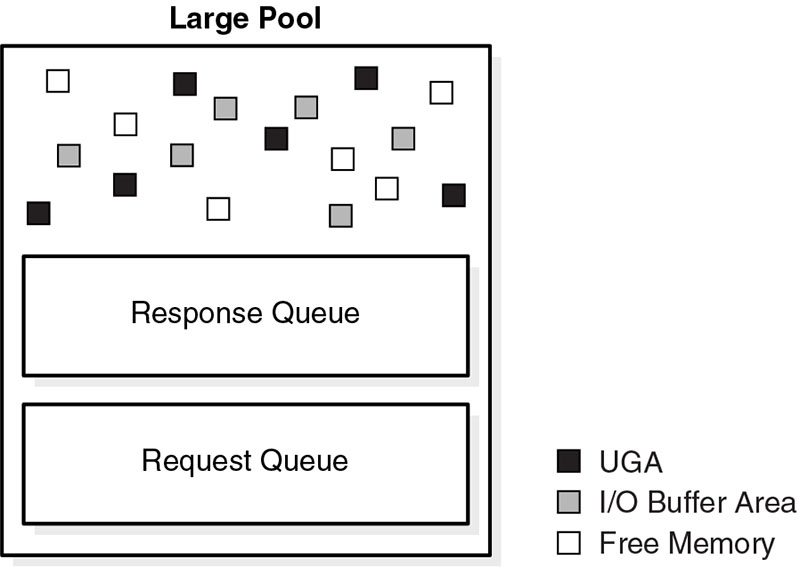
The large pool is an optional memory area intended for memory allocations that are larger than is appropriate for the shared pool. The large pool can provide large memory allocations for the following:

1. UGA for the shared server and the Oracle XA interface (used where transactions interact with multiple databases)
2. Message buffers used in the parallel execution of statements
3. Buffers for Recovery Manager (RMAN) I/O slaves

By allocating session memory from the large pool for shared SQL, the database avoids performance overhead caused by shrinking the shared SQL cache. By allocating memory in large buffers for RMAN operations, I/O server processes, and parallel buffers, the large pool can satisfy large memory requests better than the shared pool.

Werbung

Figure 5-8 is a graphical depiction of the *large pool*.

Figure 5-8 Large Pool

The large pool is different from reserved space in the shared pool, which uses the same LRU list as other memory allocated from the shared pool. The large pool does not have an LRU list. Pieces of memory are allocated and cannot be freed until they are done being used. As soon as a chunk of memory is freed, other processes can use it.

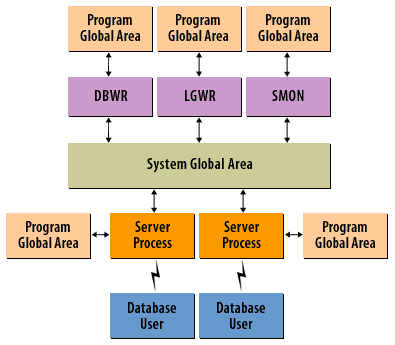
**Java Pool**

The Java pool is an area of memory that stores all session-specific Java code and data within the Java Virtual Machine (JVM). This memory includes Java objects that are migrated to the Java session space at end-of-call. For dedicated server connections, the Java pool includes the shared part of each Java class, including methods and read-only memory such as code vectors, but not the per-session Java state of each session. For shared server, the pool includes the shared part of each class and some UGA used for the state of each session. Each UGA grows and shrinks as necessary, but the total UGA size must fit in the Java pool space. The Java Pool Advisor statistics provide information about library cache memory used for Java and predict how changes in the size of the Java pool can affect the parse rate. The Java Pool Advisor is internally turned on when statistics\_level is set to TYPICAL or higher. These statistics reset when the advisor is turned off

**Each Session receives its Own Memory in Oracle**

**Private memory for Process**

A *Program Global Area*, or *PGA* as it is often called, is an area of private memory set aside for the exclusive use of one process. Each process that connects to an Oracle database, whether a *server process* or a *background process*, gets a Program Global Area assigned to it.  
The following diagram illustrates how this works, and shows the relationship between processes, the SGA, and the PGA:



1. Program Global Area: This is a private memory area for a process.
2. Program Global Area: This is a private memory area for a process.
3. Server Process: Server processes are created when users connect to Oracle.
4. System Global Area: This shared memory area used by all processes
5. DBWR, LGWR, and SMON: Even the background processes get private memory
6. Database User: Users communicate with server processes over the LAN
7. Program Global Area: This is a private memory area for a process.
8. Program Global Area: This is a private memory area for a process
9. Program Global Area: This is a private memory area for a process.

Werbung

1. Program Global Area: This is a private memory area for a process.

[**PGA DBWR SGA**](https://www.relationaldbdesign.com/database-architecture/module5/pga-dbwr-sga.php)  
Program Global Areas are used to store information that does not need to be shared, or that should not be shared, by all processes and user sessions.

**Program Global Area**

The Program Global Area is an area of memory allocated and private for one process. The configuration of the PGA depends on the connection configuration of the Oracle database: either shared server or dedicated.  
In a shared server configuration, multiple users share a connection to the database, minimizing memory usage on the server, but potentially affecting response time for user requests. In a shared server environment, the SGA holds the session information for a user instead of the PGA. Shared server environments are ideal for a large number of simultaneous connections to the database with infrequent or short-lived requests.

Werbung

In a dedicated server environment, each user process gets its own connection to the database; the PGA contains the session memory for this configuration.  
The PGA also includes a sort area. The sort area is used whenever a user request requires a sort, bitmap merge, or hash join operation.  
As of Oracle9i, the PGA\_AGGREGATE\_TARGET parameter, in conjunction with the WORKAREA\_SIZE\_POLICY initialization parameter, can ease system administration by allowing the DBA to choose a total size for all work areas and let Oracle manage and allocate the memory between all user processes. As mentioned earlier in this chapter, the parameter MEMORY\_TARGET manages the PGA and SGA memory as a whole to optimize performance.  
The next lesson talks more about this.

**DBWn**

The database writer process, known as DBWR in older versions of Oracle, writes new or changed data blocks (known as dirty blocks) in the buffer cache to the datafiles. Using an LRU algorithm, DBWn writes the oldest, least active blocks first. As a result, the most commonly requested blocks, even if they are dirty blocks, are in memory. Up to 20 DBWn processes can be started, DBW0 through DBW9 and DBWa through DBWj. The number of DBWn processes is controlled by the DB\_WRITER\_PROCESSES parameter.

**System Global Area**

The System Global Area is a group of shared memory structures for an Oracle instance, shared by the users of the database instance. When an Oracle instance is started, memory is allocated for the SGA based on the values specified in the initialization parameter file or hard-coded in the Oracle software. Many of the parameters that control the size of the various parts of the SGA are dynamic; however, if the parameter SGA\_MAX\_SIZE is specified, the total size of all SGA areas must not exceed the value of SGA\_MAX\_SIZE. If SGA\_MAX\_SIZE is not specified, but the parameter SGA\_TARGET is specified, Oracle automatically adjusts the sizes of the SGA components so that the total amount of memory allocated is equal to SGA\_TARGET. SGA\_TARGET is a dynamic parameter; it can be changed while the instance is running. The parameter MEMORY\_TARGET, new to Oracle 11g, balances all memory available to Oracle between the SGA and the Program Global Area (discussed later in this chapter) to optimize performance. Memory in the SGA is allocated in units of granules. A granule can be either 4MB or 16MB, depending on the total size of the SGA. If the SGA is less than or equal to 128MB, a granule is 4MB; otherwise, it is 16MB.

**Oracle PGA Contents**

Werbung

**The two components of PGA**

When you execute an SQL statement, the source and the execution plan are stored in shared memory because all processes can use that information. Information that is process specific, such as bind variable information and PL/SQL variable values, is contained in the PGA. Under the standard configuration of Oracle, the Program Global Area contains the following two types of information:

1. Stack space
2. Session information

**Stack Space**

The stack space is used to hold process variables, arrays, and other similar information.

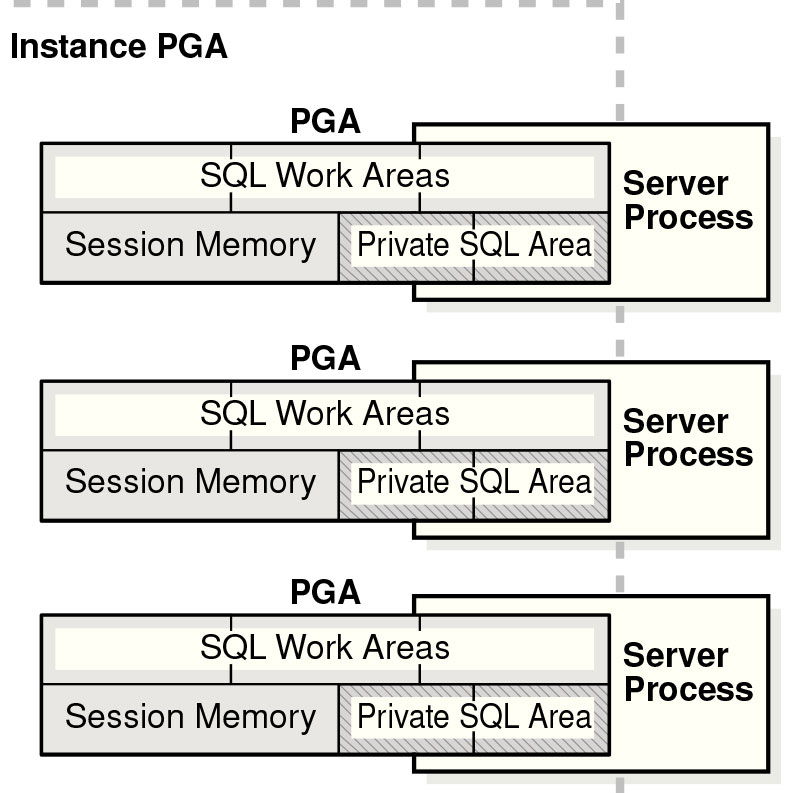
**Session information**

Session information includes PL/SQL variables and the private SQL areas. Under the multi-threaded server configuration, the session information is contained in the [**SGA**](https://www.relationaldbdesign.com/database-architecture/module5/sharedServer-pga-contents.php).

**Overview of the Program Global Area (PGA)**

The PGA is memory specific to an operating process or thread that is not shared by other processes or threads on the system. Because the PGA is process-specific, it is never allocated in the SGA. The PGA is a memory heap that contains session-dependent variables required by a dedicated or shared server process. The server process allocates memory structures that it requires in the PGA. An analogy for a PGA is a temporary countertop workspace used by a file clerk. In this analogy, the file clerk is the server process doing work on behalf of the customer (client process). The clerk clears a section of the countertop, uses the workspace to store details about the customer request and to sort the folders requested by the customer, and then gives up the space when the work is done. Figure 5-10 shows an instance PGA (collection of all PGAs) for an instance that is not configured for shared servers. You can use an initialization parameter to set a target maximum size of the instance PGA. Individual PGAs can grow as needed up to this target size.

**Instance PGA**

Figure 5-10 Instance PGA

**What is Sort Area in Oracle?**

Werbung

Sort Areas are the final memory structures that we are going to talk about in this module. A Sort Area is a large block of memory that Oracle uses to sort data when you issue a query with an ORDER BY clause. Under the default, which is a dedicated server configuration of Oracle, users will each have their own Sort Area.

**Initialization Parameters**

Two key initialization parameters control the size of the Sort Area. These are:

1. SORT\_AREA\_SIZE: The maximum size of the Sort Area.
2. SORT\_AREA\_RETAINED\_SIZE: This is a threshold that specifies the maximum amount of sort memory to retain for future sorts. When a sort is finished, any Sort Area memory in excess of this amount will be released.

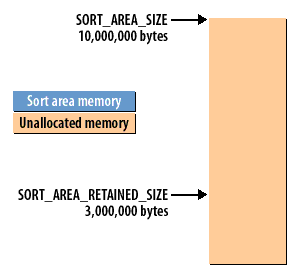
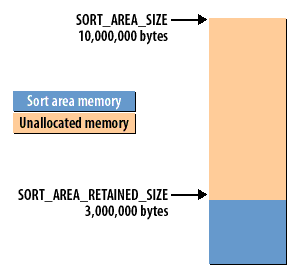
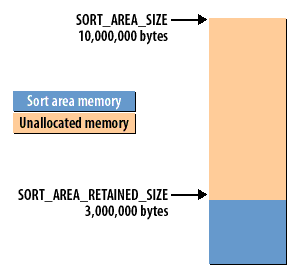
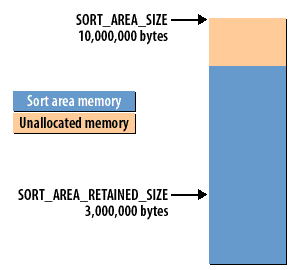
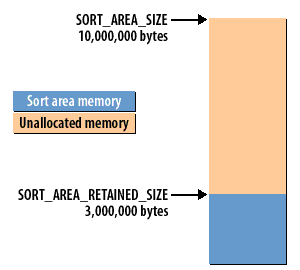
The function of the SORT\_AREA\_RETAINED\_SIZE parameter is sometimes a bit difficult to grasp. View the following slide show to see how these parameters interact:

# Distribute System Overhead evenly over Available Drives

Werbung

System overhead consists of I/O to the SYSTEM tablespace for the data dictionary, the TEMP tablespace for sorting, and the tablespaces that contain rollback segments for undo information. You should consider the system profile in spreading the system overhead over multiple drives. For example, if the application generates a lot of data changes versus data reads, the I/O to the rollback segments may increase due to higher writes for changes and higher reads for consistent read functionality.  
Sort activity can also affect disk I/O. Prior to Oracle Database 10g, youwou ld get the majority of sorts to occur in memory through tuning the SORT\_AREA\_SIZE parameter in the initialization file. Oracle constantly queries and updates the data dictionary stored in the SYSTEM tablespace, and this information is cached in the shared pool section of the SGA, so sizing your shared pool properly is a key to overall performance. As of Oracle Database 10g, Oracle can automatically and dynamically size the different pools in the SGA.

## Oracle Sort Area

1) If you expected most sorts to take less than 3,000,000 bytes, you might configure your database as shown here  
2) Small sorts would cause memory to be allocated below the retained size threshold  
3) This memory would not be released after the sort.It would remain allocated to the sort area to be used for future sorts.  
4) A large sort could consume up to the amount of memory specified by SORT\_AREA\_SIZE  
5) After a large sort, enough memory would be released to bring the sort area's size back down to the level specified by the SORT\_AREA\_RETAINED\_SIZE parameter

### Sort Activity

Sort activity can also affect disk I/O. Prior to Oracle Database 10g, youwou ld get the majority of sorts to occur in memory through tuning the SORT\_AREA\_SIZE parameter in the initialization file. Oracle constantly queries and updates the data dictionary stored in the SYSTEM tablespace, and this information is cached in the shared pool section of the SGA, so sizing your shared pool properly is a key to overall performance. As of Oracle Database 10g, Oracle can automatically and dynamically size the different pools in the SGA.

### Memory for sorting within the PGA

Each server process uses memory in its PGA for sorting rows before returning them to the user. If the memory allocated for sorting is insufficient to hold all the rows that need to be sorted, the server process sorts the rows in multiple passes called runs. The intermediate runs are written to the temporary tablespace of the user, which reduces sort performance because it involves disk I/O. Sizing the sort area of the PGA was a critical tuning point in Oracle database releases prior to Oracle Database 10g. A sort area that was too small for the typical amount of data requiring sorting would result in temporary tablespace disk I/O and reduced performance. A sort area that was significantly larger than necessary would waste memory.

Werbung

Werbung

As of Oracle Database 10g, the database provides automatic sizing for the PGA. By default, this memory management is enabled, and sizing for PGA work areas is based on 20 percent of the SGA memory size. By using automatic sizing for the PGA, you eliminate the need to size individual portions of the PGA, such as SORT\_AREA\_SIZE.  
Oracle Database 11g introduced automatic memory management that spans both the SGA and the PGA. By setting a single MEMORY\_TARGET initialization parameter (given that the PGA size can be based on a percentage of the SGA memory size), the PGA and SGA will be automatically set to appropriate initial values. Oracle then tunes memory for optimal SGA and PGA performance on an ongoing basis.

### Sizing the Sort Area

Sorts that can be processed entirely in memory execute faster than those that cannot.If you find performance suffering from a lot of disk-based sorts, one remedy to consider would be to increase the Sort Area size. Remember though, that each user potentially could end up with SORT\_AREA\_RETAINED\_SIZE bytes allocated to them. Be sure to consider the impact of that in terms of how much memory is available on the system.

Werbung

Werbung

Space for Sort Areas actually comes from the PGA, and using the [**Shared Server Sort Areas**](https://www.relationaldbdesign.com/database-architecture/module5/sharedServer-sort-areas.php) changes how some of that memory is allocated.  
Oracle does not recommend using the SORT\_AREA\_SIZE parameter unless the instance is configured with the shared server option. Oracle recommends that you enable automatic sizing of SQL working areas by setting PGA\_AGGREGATE\_TARGET instead. SORT\_AREA\_SIZE is retained for backward compatibility.

**Understanding Oracle's Memory Architecture Conclusion**

This module discussed how Oracle uses Sort Areas, shrinking and expanding them as necessary, depending on how much data there is to sort.  
Now you should be able to:

1. Describe the memory structures that make up the SGA
2. Describe how the database buffer cache manages data blocks
3. Describe the purpose of the three types of buffer pools
4. Describe the function of the redo log buffer.
5. Identify the function of the large pool
6. Describe how each session gets its own memory
7. Name the contents of the PGA.
8. Describe how Oracle uses memory for sorting

**Glossary**

This module introduced you to the following terms:

1. *background processes*
2. *dedicated server process:A database configuration in which a server process handles requests for a single client process.*
3. *default buffer pool:*Placing a table into the keep pool merely changes the part of the buffer cache where the blocks are stored. Instead of the blocks being cached in the default buffer pool, they are cached in the keep buffer pool. No separate algorithm is used to control keep pool caching.
4. *dirty list:* The dirty list points to all the buffers that have been modified and that need to be written back to disk. Sometimes dirty buffers will be found in the LRU List as well, but eventually they too will make it to the dirty list.
5. *execution plan*
6. *keep buffer pool: An alternate buffer pool where by convention you would assign segments that were accessed fairly frequently, but still got aged out of the default buffer pool due to other segments needing space.*
7. *large pool:* Optional area in the SGA that provides large memory allocations for backup and restore operations, I/O server processes, and session memory for the shared server and Oracle XA.
8. *Program Global Area:* The Program Global Area (PGA) is a private memory region that contains the data and control information for a server process. Only a server process can access the PGA. Oracle Database reads and writes information in the PGA on behalf of the server process.
9. *parsing: The process of pulling apart a SQL statement and figuring out how best to execute it.*
10. *PGA:*
11. *recycle buffer pool:* A pool of buffers in the database buffer cached used for that should be held in memory for the minimum amount of time possible.
12. *server process:* A process on the server that communicates with a user process, and which processes SQL statements submitted by that user process.
13. *Sort Area:* SORT\_AREA\_SIZE specifies (in bytes) the maximum amount of memory Oracle will use for a sort. After the sort is complete, but before the rows are returned, Oracle releases all of the memory allocated for the sort, except the amount specified by the SORT\_AREA\_RETAINED\_SIZE parameter.
14. *System Global Area:* In the database management systems developed by the Oracle Corporation, the System Global Area (SGA) forms the part of the system memory (RAM) shared by all the processes belonging to a single Oracle database instance. The SGA contains all information necessary for the instance operation.

**Introduction to Database Creation and Preparation**

**Preparing to create a database**

Finally, you have a chance to do some real work with Oracle. In this module, we will make preparations to create the database that will be used for the remainder of this course. Together, we will:

1. Review Oracle's Optimal Flexible Architecture
2. Create a database parameter file
3. Size the SGA and the database buffer cache
4. Decide on a database block size
5. Size the initial database files

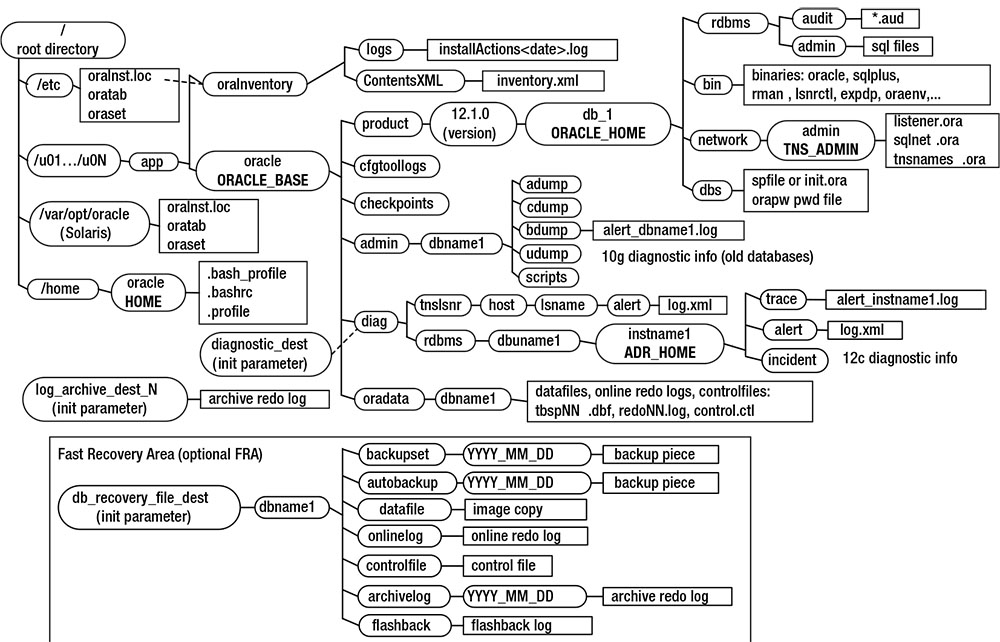
In the module following this one, we will actually create the course project database that we discussed earlier in the course. If you are running Oracle, you'll be building an actual database, and if you are not running Oracle, we'll walk you through the steps so you know how to do it. In addition to being used for this course, this project database will also be used for the next four courses in the *Oracle Database Administration Certification Series*. Now, let us make the necessary preparations to create a database.

**Understanding the OFA**

Before you install Oracle and start creating databases, you must understand Oracle's Optimal Flexible Architecture (OFA) standard. This standard is widely employed for specifying consistent directory structures and the file-naming conventions used when installing and creating Oracle databases.

Werbung

Note One irony of this ubiquitous OFA "standard" is that almost every DBA, in some manner, customizes it to fit the unique requirements of his or her environment.  
Because most shops implement a form of the OFA standard, understanding this structure is critical. Figure 6-1 shows the directory structure and file names used with the OFA standard.  
Not all the directories and files found in an Oracle environment appear in this figure (there is not enough room). However, the critical and most frequently used directories and files are displayed.

Figure 6-1. Oracle's OFA standardThe OFA standard includes several directories that you should be familiar with:

1. Oracle inventory directory
2. Oracle base directory (ORACLE\_BASE)
3. Oracle home directory (ORACLE\_HOME)
4. Oracle network files directory (TNS\_ADMIN)
5. Automatic Diagnostic Repository (ADR\_HOME)

These directories are discussed in the following sections

**Database Environment**

If you intend to work through the process of creating a database, there are some things that you should make sure of before you proceed through this module. Some of you may be taking this course without access to the Oracle software. In that case, you do not need to worry about any of the preperation described in this lesson. You should still work through this module. It contains important information that you need to know as a DBA. Assuming that you do intend to work through the process of creating a database, please make sure of the following:

1. If you haven't yet installed the Oracle database software on your system, you should do that now.
2. Be sure that you know the passwords for the SYSTEM and SYS users
3. Make sure you have access to a text editor such as vi (UNIX) or NotePad (Windows NT), and that you know how to use it.

The easiest way to prepare for this module would be to get a copy of Oraclei for Windows NT and install it, taking all the default options. The Windows NT installation is very easy to do, as compared with UNIX, and using Windows NT should enable you to use some machine other than your production server for this course. If you want to use UNIX, you will need an experienced DBA to help you install the software, and to help you set up the proper access and permissions that you need in order to create a database.  
Before continuing, be sure you have completed the necessary steps listed above.

**Oracle Inventory Directory**

The Oracle inventory directory stores the inventory of Oracle software installed on the server. This directory is required and is shared among all installations of Oracle software on a server. When you first install Oracle, the installer checks to see whether there is an existing OFA-compliant directory structure in the format /u[01-09]/app. If such a directory exists, then the installer creates an Oracle inventory directory, such as

/u01/app/oraInventory

If the ORACLE\_BASE variable is defined for the oracle operating system (OS) user, then the installer creates a directory for the location of Oracle inventory, as follows:

ORACLE\_BASE/../oraInventory

For example, if ORACLE\_BASE is defined as /ora01/app/oracle, then the installer defines the location of Oracle inventory as

/ora01/app/oraInventory

If the installer does not find a recognizable OFA-compliant directory structure or an ORACLE\_BASE variable, then the location for Oracle inventory is created under the HOME directory of the oracle user. For instance, if the HOME directory is /home/oracle, then the location of Oracle inventory is

Ad

/home/oracle/oraInventory

## Why should I use (OFA) Optimal Flexible Architecture

Optimal Flexible Architecture is a set of guidelines and naming conventions, nothing more. You do not have to follow it, and many sites do deviate to some degree or another. I suppose that without too much trouble, someone could dream up a different but equally flexible set of conventions for placing and naming the directories and files used in an Oracle installation. I think one of the biggest benefits of OFA is that it has been so widely adopted. Once you get used to it, you will find that you can move from one client to the next, and immediately be productive because everything is set up in pretty much the same way. There is a lot to be said for that.

OFA defines where each component will install its files. In many respects it resembles the Filesystem Hierarchy Standard (FHS) on UNIX systems in which, for example, the directory "/bin" always holds the essential system binaries — so when administrators and users use other systems they will already know where to find the standard system binaries. OFA takes FHS standard-style concepts and uses them for Oracle products on UNIX and on Windows. In this way Oracle Database administrators will find familiar structures and locations of the various applications and data installed on any OFA-compliant system. OFA covers where to install each part of each product; it addresses the storage of both applications and data. Much like the FHS, OFA imposes no constraints on the locations: it merely makes recommendations. Oracle Corporation has structured the OFA system so that system administrators can use multiple disks (for example: applications on one disk and databases on another). OFA also allows for installing multiple versions of the same product on the same host: for example Oracle Database 9 and Oracle Database 10.

### Oracle Base Directory

The Oracle base directory is the topmost directory for Oracle software installation. You can install one or more versions of the Oracle software beneath this directory. The OFA standard for the Oracle base directory is as follows:

/<mount\_point>/app/<software\_owner>

Typical names for the mount point include /u01, /ora01, /oracle, and /oracle01. You can name the mount point according to whatever your standard is for your environment. I prefer to use a mount-point name such as /ora01. It is short, and when I look at the mount points on a database server, I can immediately tell which are used for the Oracle database. Also, a short mount-point name is easier to use when you are querying the data dictionary to report on the physical aspects of your database. Additionally, a shorter mount-point name makes for less typing when you are navigating through directories via OS commands. The software owner is typically named oracle. This is the OS user you use to install the Oracle software (binaries). Listed next is an example of a fully formed Oracle base directory path:

/u01/app/oracle

### Oracle Home Directory

The Oracle home directory defines the installation location of software for a particular product, such as Oracle Database 12c or Oracle Database 11g. You must install different products or different releases of a product in separate Oracle homes. The recommended OFA-compliant Oracle home directory is as follows:

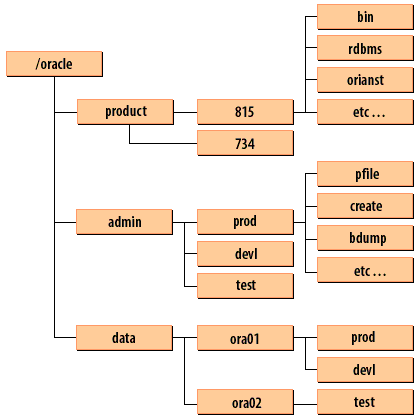
ORACLE\_BASE/product//<version>//<install\_name>

In the previous line of code, possible versions include 12.1.0.1 and 11.2.0.3. Possible install\_name values include db\_1, devdb1, test2, and prod1. Here is an example of an Oracle home name for a 12.1 database:

/u01/app/oracle/product/12.1.0.1/db\_1

**Note:**Some DBAs dislike the db\_1 string on the end of the ORACLE\_HOME directory and see no need for it. The reason for the db\_1 is that you may have two separate installations of binaries: a development installation and a test installation. If you do not require that configuration in your environment, feel free to drop the extra string (db\_1).

Oracle Corporation offers specific recommendations on the naming and placement of database files to help you separate the different types of files in an Oracle installation.  
This eases administration and spreads I/O across as many devices as possible. The Oracle database software, the control files and the datafiles, and the administrative files are all stored in separate directory trees and can be moved around independently of each other. Oracle refers to their recommendations as the *Optimal Flexible Architecture (OFA)*. Following OFA creates a directory structure like the one below. Move your mouse over the following diagram for more explanations.

/oracle --> product

1. oracle: The top-level directory is called the Oracle Base directory. On UNIX systems, the ORACLE\_BASE environment variable will point to this. Windows uses a registry entry named ORACLE\_BASE to point to this directory.
2. product: Software for Oracle products is installed under this directory.
3. 815: Multiple releases of Oracle may be installed on the same machine. The first three digits of the release number form the directory name. This directory contains the Oraclei 8.1.5 release. Using separate directories for each release enables you to install a new release, or delete an old one, without affecting the others. These directories are referred to as Oracle Homes.
4. 734: This directory would contain the 7.3.4 release of Oracle, and would also contain subdirectories named bin, rdbms, orainst, and so forth.
5. bin, rdbms, orainst, etc.: A number of different directories are used to contain various elements of the Oracle database software.
6. admin: This directory tree contains administrative files such as scripts, parameter files, and database alert logs.
7. prod, devl, test: Each Oracle database gets its own subdirectory underneath the admin directory.
8. pfile: Contains parameter files for the prod database.
9. create: Contains database creation scripts for the prod database.
10. bdump: Contains the alert log and trace files for the prod database.
11. data: Under UNIX, you can place all the mount points used for data files under the data directory. Not all sites do this however. Under Windows NT you can't do it at all, because that would force you to put all database files on the same physical disk.
12. ora01, ora02: Mount points used for database files are usually numbered, and are also named to make it obvious that they contain Oracle database files.
13. prod: This directory contains files for the production database.
14. devl: This directory contains files for the development database.
15. test: This directory contains files for the test database.

**Using Oracle Managed Files**

The Oracle Managed File (OMF) feature automates many aspects of tablespace management, such as file placement, naming, and sizing. You control OMF by setting the following initialization parameters:

1. DB\_CREATE\_FILE\_DEST

Werbung

1. DB\_CREATE\_ONLINE\_LOG\_DEST\_N
2. DB\_RECOVERY\_FILE\_DEST

If you set these parameters before you create the database, Oracle uses them for the placement of the data files, control files, and online redo logs. You can also enable OMF after your database has been created. Oracle uses the values of the initialization parameters for the locations of any newly added files. Oracle also determines the name of the newly added file. The advantage of using OMF is that creating tablespaces is simplified. For example, the CREATE TABLESPACE statement does not need to specify anything other than the tablespace name.  
First, enable the OMF feature by setting the DB\_CREATE\_FILE\_DEST parameter:

SQL> alter system set db\_create\_file\_dest='/u01';

Now, issue the CREATE TABLESPACE statement:

SQL> create tablespace inv1;

This statement creates a tablespace named INV1, with a default data file size of 100MB. Keep in mind that you can override the default size of 100MB by specifying a size:

Werbung

SQL> create tablespace inv2 datafile size 20m;

To view the details of the associated data files, query the V$DATAFILE view, and note that Oracle has created subdirectories beneath the /u01 directory and named the file with the OMF format:

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

NAME

------------------------------------------------------------

/u01/O12C/datafile/o1\_mf\_inv1\_8b5l63q6\_.dbf

/u01/O12C/datafile/o1\_mf\_inv2\_8b5lflfc\_.dbf

One limitation of OMF is that you are limited to one directory for the placement of data files. If you want to add data files to a different directory, you can alter the location dynamically:

SQL> alter system set db\_create\_file\_dest='/u02';

Werbung

Although this procedure is not a huge deal, I find it easier not to use OMF. Most of the environments I have worked in have many mount points assigned for database use.  
You do not want to have to modify an initialization parameter every time you need a data file added to a directory that is not in the current definition of DB\_CREATE\_FILE\_DEST.  
It is easier to issue a CREATE TABLESPACE statement or ALTER TABLESPACE statement that has the file location and storage parameters in the script. It is not cumbersome to provide directory names and file names to the tablespace-management statements.

[Oracle Base Directory](https://www.relationaldbdesign.com/database-architecture/module6/oracle-base-directory.php)  
There is more to the Optimal Flexible Architecture than just the directory structure shown here. We will be following the OFA guidelines as we create the course database, and I'll be pointing out other features of the OFA as we go along.