|  |
| --- |
| EPAM Systems, RD Dep. |
| MTN.BI.07 Oracle Database Architecture |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | [Kiryl Bucha](mailto:Kiryl_Bucha@epam.com) | 12-JAN-2012 |  |  |
| 2.0 | Updated in accordance with renewed content | [Elias Nema](mailto:Elias_Nema@epam.com) | 20-JAN-2014 |  |  |

Contents

[1. Physical components 3](#_Toc383286942)

[1.1. Control Files 3](#_Toc383286943)

[1.2. Datafiles 3](#_Toc383286944)

[1.2.1. Setting the Database Block Size 3](#_Toc383286945)

[1.2.2. Datafile structure 4](#_Toc383286946)

[1.2.3. Extents and segments 4](#_Toc383286947)

[1.2.4. Storage Hierarchy Summary 4](#_Toc383286948)

[1.3. Redo Logfiles 4](#_Toc383286949)

[1.3.1. Suppressing Redo Logging 5](#_Toc383286950)

[1.3.2. Multiplexing redo logfiles 5](#_Toc383286951)

[1.3.3. How Oracle uses the redo logs 5](#_Toc383286952)

[1.3.4. Archived redo logs 5](#_Toc383286953)

[2. Instance Memory and Processes 6](#_Toc383286954)

[2.1. System Global Area 7](#_Toc383286955)

[2.1.1. Database buffer cache 7](#_Toc383286956)

[2.1.2. Shared pool 8](#_Toc383286957)

[2.1.3. Redo log buffer 8](#_Toc383286958)

[2.1.4. Other pools in the SGA 8](#_Toc383286959)

[2.2. Background Processes for an Instance 8](#_Toc383286960)

[2.3. Connecting to Oracle 9](#_Toc383286961)

[2.3.1. Server Processes and Clients 9](#_Toc383286962)

[2.3.2. Server process 10](#_Toc383286963)

[2.3.3. Client process 10](#_Toc383286964)

[2.3.4. Dedicated Server 10](#_Toc383286965)

[2.3.5. Shared Server 11](#_Toc383286966)

[2.4. Mechanics of Connecting over TCP/IP 12](#_Toc383286967)

[2.5. The Process Global Area and User Global Area 13](#_Toc383286968)

[3. SQL Execution: 14](#_Toc383286969)

[4. Source Books and Articles 14](#_Toc383286970)

# Physical components

Three principal database file types are:

* Data files: These are for the database; they hold your tables, indexes, and all other data segment types.
* Control files: These tell you where the data files, temp files, and redo log files are, as well as other relevant metadata about their state. They also contain backup information maintained by RMAN (Recovery Manager, the backup and recovery tool).
* Redo log files: These are your transaction logs.

## Control Files

A database should have at least two copies of the control file on different physical disks. Without a current copy of the control file, you run the risk of losing track of portions of your database. Losing control files is not necessarily fatal—there are ways to rebuild them. However, rebuilding control files can be difficult, introduces risk, and can be easily avoided.

The location of the control files is defined by the CONTROL\_FILES initialization parameter. You can specify multiple copies of control files by indicating multiple locations in the CONTROL\_FILES parameter for the instance, as illustrated here:

control\_files = (/u00/oradata/control.001.dbf,

/u01/oradata/control.002.dbf,

/u02/oradata/control.003.dbf)

This parameter tells the instance where to find the control files. Oracle will ensure that all copies of the control file are kept in sync so all updates to the control files will occur at the same time. If you do not specify this parameter, Oracle will create a control file using a default filename or by leveraging Oracle Managed Files (if enabled).

Control files are something a developer will probably never have to actually deal with. To a DBA, they are an important part of the database, but to a software developer they are not relevant. You do not need to be concerned with additional performance impact when writing to multiple control files. Updates to the control files are insignificant compared to other disk I/O that occurs in an Oracle environment.

## Datafiles

Datafiles contain the actual data stored in the database, the tables and indexes that store data, the data dictionary that maintains information about these data structures, and the rollback segments used to implement multiuser concurrency.

A datafile is composed of Oracle Database blocks that, in turn, are composed of operating system blocks on a disk. Oracle block sizes range from 2 KB to 32 KB. Prior to Oracle9i, only a single block size could be present in the entire database. In versions of the database since the introduction of Oracle9i, you still set a default block size for the database, but you can also have up to five other block sizes in a database (though only a single block size for each tablespace).

Datafiles belong to only one database and to only one tablespace within that database. Data is read in units of Oracle blocks from the datafiles into memory as needed, based on the work users are doing. Blocks of data are written from memory to the datafiles stored on disk as needed to ensure that the database reliably records changes made by users.

### Setting the Database Block Size

Prior to Oracle9i, you set the database block size for an Oracle Database at the time you created the database, and you couldn’t change it without re-creating the database. Since Oracle9i, you have more flexibility, because you can have multiple block sizes in the same database. In all versions, the default block size for the database is set using the DB\_BLOCK\_SIZE instance initialization parameter.

How do you choose an appropriate block size for an Oracle Database? Oracle defaults to a block size based on the operating system used, but understanding the implications of the block size can help you determine a more appropriate setting for your workload. The block size is the minimum amount of data that can be read or written at one time. In online transaction processing (OLTP) systems, a transaction typically involves a relatively small, well-defined set of rows, such as the rows used for placing an order for a set of products for a specific customer. The access to rows in these operations tends to be through indexes, as opposed to through a scan of the entire table. Because of this, having smaller blocks (4 KB) might be appropriate. Oracle won’t waste system resources by accessing larger blocks that contain additional data not required by the transaction.

Data warehouses workloads can include reading millions of rows and scans of all the data in a table. For this type of activity, using bigger database blocks enables each block read to deliver more data to the requesting user. To support these operations best, data warehouses usually have larger blocks, such as 8 KB or 16 KB. Each I/O operation might take a little longer due to the larger block size, but the reduced number of operations will end up improving overall performance.

### Datafile structure

The first block of each datafiles called the datafile header. It contains critical information used to maintain the overall integrity of the database. One of the most critical pieces of information in this header is the checkpoint structure. This is a logical timestamp that indicates the last point at which changes were written to the datafile. This timestamp is critical during an Oracle recovery process as the timestamp in the header determines which redo logs to apply in bringing the datafile to the current point in time.

### Extents and segments

From a physical point of view, a datafile is stored as operating system blocks. From a logical point of view, datafiles have three intermediate organizational levels: data blocks, extents, and segments. An extent is a set of data blocks that are contiguous within an Oracle datafile. A segment is an object that takes up space in an Oracle Database, such as a table or an index that is composed of one or more extents. Segments are the major organizational structure within a tablespace. Segments are simply your database objects that consume storage—typically objects such as tables, indexes, undo segments, and so on. Most times, when you create a table, you create a table segment. When you create a partitioned table, you are not creating a table segment, rather you create a segment per partition. When you create an index, you normally create an index segment, and so on. Every object that consumes storage is ultimately stored in a single segment. There are undo segments, temporary segments, cluster segments, index segments, and so on.

When Oracle updates data, it first attempts to update the data in the same data block. If there is not enough room in the data block for the new information, Oracle will write the data to a new data block that could be in a different extent. This discussion is especially important if you are running an older release of Oracle. Oracle Database 10g added a Segment Advisor that greatly simplifies reclaiming unused space in current database versions.

### Storage Hierarchy Summary

In summary, the hierarchy of storage in Oracle is as follows:

1. **A database** is made up of one or more tablespaces.
2. **A tablespace** is made up of one or more data files. These files might be cooked files in a file system, raw partitions, ASM managed database files, or a file on a clustered file system. A tablespace contains segments.
3. **A segment** (TABLE, INDEX, and so on) is made up of one or more extents. A segment exists in a tablespace, but may have data in many data files within that tablespace.
4. **An extent** is a logically contiguous set of blocks on disk. An extent is in a single tablespace and, furthermore, is always in a single file within that tablespace.
5. **A block** is the smallest unit of allocation in the database. A block is the smallest unit of I/O used by a database.

## Redo Logfiles

Redo logfiles contain a “recording” of the changes made to the database as a result of transactions and internal Oracle activities. Since Oracle usually caches changed blocks in memory, when instance failure occurs, some changed blocks might not have been written out to the datafiles. The recording of the changes in the redo logs can be used to play back the changes lost when the failure occurred, thus protecting transactional consistency.

***These files are sometimes confused with rollback buffers supporting concurrency. They are not the same!***

In addition, redo logfiles are used for “undo” operations when a ROLLBACK statement is issued. Uncommitted changes to the database are rolled back to the database image at the last commit. In addition to using the NOLOGGING keyword in certain commands, you can also mark a table or an entire tablespace with the NOLOGGING attribute. This will suppress redo information for all applicable operations on the table or for all tables in the tablespace.

### Suppressing Redo Logging

By default, Oracle logs all changes made to the database. The generation of redo logs adds a certain amount of overhead. You can suppress redo log generation to speed up specific operations, but doing so means the operation in question won’t be logged in the redo logs and you will not be able to recover that operation in the event of a failure. If you do decide to suppress redo logging for certain operations, you would include the NOLOGGING keyword in the SQL statement for the operation. If a failure occurred during the operation, you would need to repeat the operation. For example, you might build an index on a table without generating redo information. In the event that a database failure occurs and the database is recovered, the index will not be re-created because it wasn’t logged. You’d simply execute the script originally intended to create the index again.

### Multiplexing redo logfiles

Oracle defines specific terminology to describe how it manages redo logs. Each Oracle instance uses a thread of redo to record the changes it makes to the database. A thread of redo is composed of redo log groups, which are composed of one or more redo log members.

Logically, you can think of a redo log group as a single redo logfile. However, Oracle allows you to specify multiple copies of a redo log to protect the all-important integrity of the redo log. By creating multiple copies of each redo logfile, you protect the redo logfile from disk failure and other types of disasters.

When multiple members are in a redo log group, Oracle maintains multiple copies of the redo logfiles. The same arguments used for multiplexing of control files apply here. However, though you can rebuild the static part of a control file if you lose it, there is no way to reproduce a lost redo logfile. So, be sure to have multiple copies of the redo file. Simple redundant disk protection is not sufficient for cases in which human error results in the corruption or deletion of a redo logfile.

Oracle writes synchronously to all redo log members. Oracle will wait for confirmation that all copies of the redo log have been successfully updated on disk before the redo write is considered done. If you put one copy on a fast or lightly loaded disk, and one copy on a slower or busier disk, your performance will be constrained by the slower disk. Oracle has to guarantee that all copies of the redo logfile have been successfully updated to avoid losing data.

Consider what could happen if Oracle were to write multiple redo logs asynchronously, writing to a primary log and then updating the copies later in the background. If a failure occurs that brings the system down and damages the primary log, Oracle might not have completed updating all the logs. At this point you have committed transactions that are lost—the primary log that recorded the changes made by the transactions is gone, and the copies of the log are not yet up to date with those changes. To prevent this from occurring, Oracle always waits until all copies of the redo log have been updated.

### How Oracle uses the redo logs

Once Oracle fills one redo logfile, it automatically begins to use the next logfile. When the server cycles through all the available redo logfiles, it returns to the first one and reuses it. Every Oracle database has at least two online redo log file groups. The act of switching from one log file group to another is called a log switch. It is important to note that a log switch may cause a temporary “pause” in a poorly configured database. Since the redo logs are used to recover transactions in the event of a failure, we must be certain we won’t need the contents of a redo log file before we are able to overwrite it.

### Archived redo logs

You may be wondering how to avoid losing the critical information in the redo log when Oracle cycles over a previously used redo log.

There are actually two ways to address this. The first is quite simple: you don’t avoid losing the information and you suffer the consequences in the event of a failure. You will lose the history stored in the redo file when it is overwritten. If a failure occurs that damages the datafiles, you must restore the entire database to the point in time when the last backup occurred. Since no redo log history exists to reproduce the changes made since the last backup occurred, you will lose the effects of those changes.

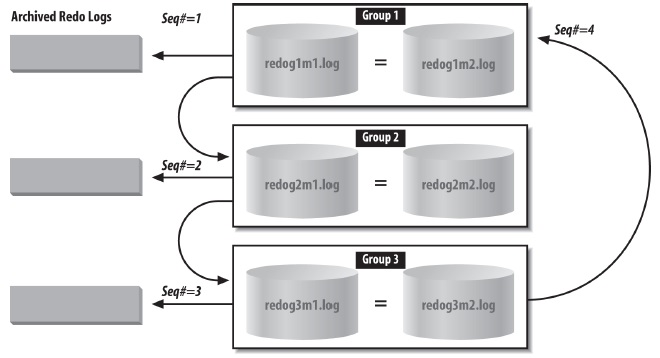
The second and more practical way to address the issue is to archive the redo logs as they fill. To understand archiving redo logs, you must first understand that there are actually two types of redo logs for Oracle:

* Online redo logs. The operating system files that Oracle cycles through to log the changes made to the database
* Archived redo logs Copies of the filled online redo logs made to avoid losing redo data as the online redo logs are overwritten

An Oracle Database can run in one of two modes with respect to archiving redo logs:

* ***NOARCHIVELOG.***As the name implies, no redo logs are archived. As Oracle cycles through the logs, the filled logs are reinitialized and overwritten, which erases the history of the changes made to the database. This mode essentially has the disadvantage mentioned above, where a failure could lead to unrecoverable data. Choosing not to archive redo logs significantly reduces your options for database backups, as we’ll discuss in Chapter 11, and is not advised by Oracle.
* ***ARCHIVELOG.*** When Oracle rolls over to a new redo log, it archives the previous redo log. To prevent gaps in the history, a given redo log cannot be reused until it is successfully archived. The archived redo logs, plus the online redo logs, provide a complete history of all changes made to the database. Together, they allow Oracle to recover all committed transactions up to the exact time a failure occurred. Operating in this mode enables tablespace and datafile backups.

The internal sequence numbers discussed earlier act as the guide for Oracle while it is using redo logs and archived redo logs to restore a database.

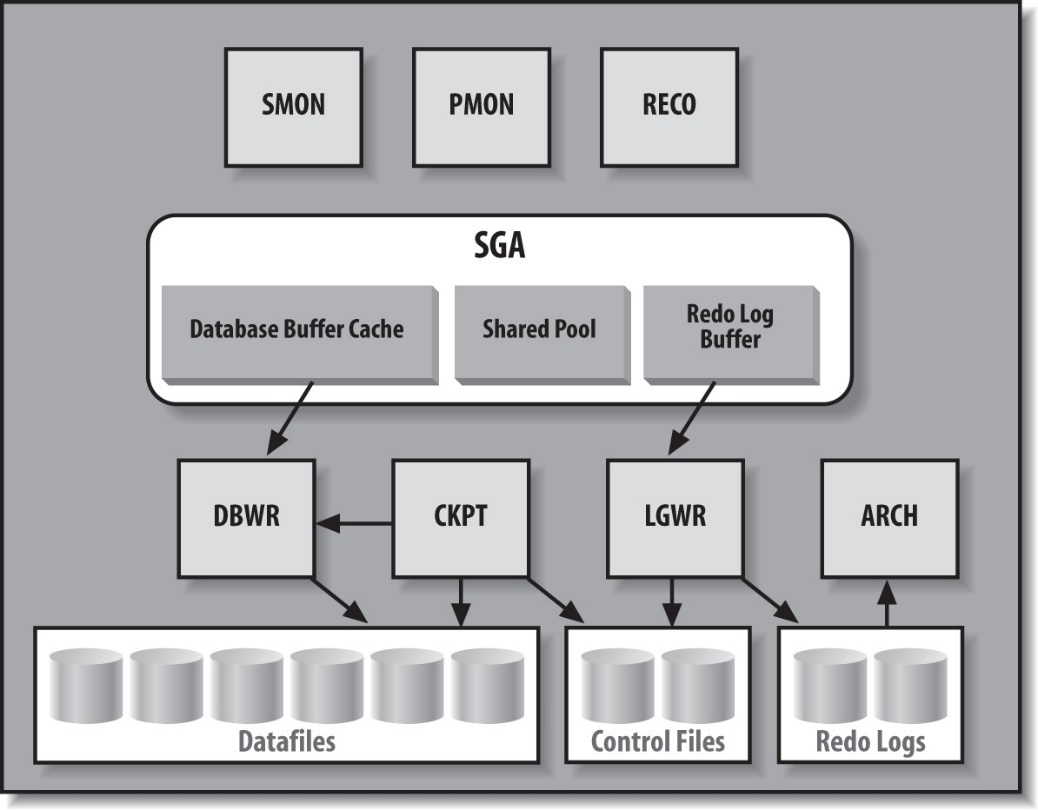


**Figure 1 Oracle Redo Logs with Archiving**

# Instance Memory and Processes

An Oracle instance can be defined as an area of shared memory and a collection of background processes. The area of shared memory for an instance is called the System Global Area, or SGA. The SGA is not really one large undifferentiated section of memory—it’s made up of various components that we’ll examine in the next section. All the processes of an instance—system processes and user processes—share the SGA. There is one SGA for an Oracle instance.

The background processes interact with the operating system and each other to manage the memory structures for the instance. These processes also manage the actual database on disk and perform general housekeeping for the instance. Figure 2 illustrates the memory structures and background processes discussed in the following section.



**Figure 2 Main Oracle SGA Structures and Background Processes**

## System Global Area

As shown in Figure 2, the System Global Area is composed of multiple areas. These include a database buffer cache, a shared pool, and a redo log buffer as shown in the figure, and also possibly a Java pool, a large pool, and a Streams pool. The following sections describe these areas of the SGA.

### Database buffer cache

The database buffer cache holds blocks of data retrieved from the database. This buffer between the users’ requests and the actual datafiles improves the performance of the Oracle Database. If a piece of data can be found in the buffer cache (for example, as the result of a recent query), you can retrieve it from memory without the overhead of having to go to disk. Oracle manages the cache using a least recently used (LRU) algorithm. If a user requests data that has been recently used, the data is more likely to be in the database buffer cache; data in the cache can be delivered immediately without a disk-read operation being executed.

When a user wants to read a block that is not in the cache, the block must be read and loaded into the cache. When a user makes changes to a block, those changes are made to the block in the cache and a record of the change is written to the redo logfile. At some later time, those changes will be written to the datafile in which the block resides. This avoids making users wait while Oracle writes their changed blocks to disk. This notion of waiting to perform I/O until absolutely necessary is common throughout Oracle. Disks are the slowest component of a computer system, so the less I/O performed, the faster the system runs. By deferring noncritical I/O operations instead of performing them immediately, an Oracle Database can deliver better performance.

The database buffer cache can be configured with buffer pools of the following types:

* DEFAULT. The standard Oracle Database buffer cache. All objects use this cache unless otherwise indicated.
* KEEP. For frequently used objects you wish to cache.
* RECYCLE. For objects you’re less likely to access again.

Both the KEEP and RECYCLE buffer pools remove their objects from consideration by the LRU algorithm.

### Shared pool

The shared pool is one of the most critical pieces of memory in the SGA, especially with regard to performance and scalability. A shared pool that is too small can kill performance to the point that the system appears to hang. A shared pool that is too large can have the same effect. A shared pool that is used incorrectly will be a disaster as well.

What exactly is the shared pool? The shared pool is where Oracle caches many bits of "program" data. When we parse a query, the parsed representation is cached there. Before we go through the job of parsing an entire query, Oracle searches the shared pool to see if the work has already been done. PL/SQL code that you run is cached in the shared pool, so the next time you run it, Oracle doesn't have to read it in from disk again. PL/SQL code is not only cached here, it is shared here as well. If you have 1,000 sessions all executing the same code, only one copy of the code is loaded and shared among all sessions. Oracle stores the system parameters in the shared pool. The data dictionary cache (cached information about database objects) is stored here. In short, everything but the kitchen sink is stored in the shared pool.

### Redo log buffer

The redo log buffer caches redo information until it is written to the physical redo logfiles stored on a disk. This buffer also improves performance (since a memory-to-memory transfer is much faster than a memory-to-disk transfer). 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 data will not reside in the redo buffer for very long. In fact, LGWR initiates a flush of this area in one of the following scenarios:

* Every three seconds
* Whenever someone commits
* When LGWR is asked to switch log files
* When the redo buffer gets one-third full or contains 1MB of cached redo log data

### Other pools in the SGA

The SGA includes several other pools:

#### 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. It is so named because it is used for allocations of large pieces of memory that are bigger than the shared pool is designed to handle. Prior to the introduction of the large pool in Oracle 8.0, all memory allocation took place in the shared pool. However this leaded to poor performance because large memory allocations tend to get a chunk of memory, use it, and then be done with it. There was no need to cache this memory.

What Oracle needed was something similar to the recycle and keep buffer pools implemented for the block buffer cache, and that’s exactly what the large pool and shared pool are now. The large pool is a recycle-style memory space, whereas the shared pool is more like the keep buffer pool—if people appear to be using something frequently, then you keep it cached.

#### Java pool

Provides memory allocation for Java objects and Java execution, including data in the Java Virtual Machine in the database.

## Background Processes for an Instance

The most common background processes are shown in Figure 2 and vary from Oracle release to release. Among the background processes in Oracle Database 12c are the following:

* **Database Writer (DBWn).** Writes database blocks from the database buffer cache in the SGA to the datafiles on disk. An Oracle instance can have up to 20 DBW processes to handle the I/O load to multiple datafiles—hence the notation DBWn. Most instances run one DBW. DBW writes blocks out of the cache for two main reasons:

If Oracle needs to perform a checkpoint (i.e., to update the blocks of the datafiles so that they “catch up” to the redo logs). Oracle writes the redo for a transaction when it’s committed, and later writes the actual blocks. Periodically, Oracle performs a checkpoint to bring the datafile contents in line with the redo that was written out for the committed transactions.

If Oracle needs to read blocks requested by users into the cache and there is no free space in the buffer cache, the blocks written out are the least recently used blocks. Writing blocks in this order minimizes the performance impact of losing them from the buffer cache.

* **Log Writer (LGWR).** Writes the redo information from the log buffer in the SGA to all copies of the current redo logfile on disk. As transactions proceed, the associated redo information is stored in the redo log buffer in the SGA. When a transaction is committed, Oracle makes the redo information permanent by invoking the Log Writer to write it to disk.
* **System Monitor (SMON).** Maintains overall health and safety for an Oracle instance. SMON performs crash recovery when the instance is started after a failure and coordinates and performs recovery for a failed instance when you have more than one instance accessing the same database, as with Real Application Clusters. SMON also cleans up adjacent pieces of free space in the datafiles by merging them into one piece and gets rid of space used for sorting rows when that space is no longer needed.
* **Process Monitor (PMON).** Watches over the user processes that access the database. 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.
* **Archiver (ARCn).** Reads the redo logfiles once Oracle has filled them and writes a copy of the used redo logfiles to the specified archive log destination(s). Up to 10 Archiver processes are possible—hence the notation ARCn. LGWR will start additional Archivers as needed, based on the load, up to a limit specified by the initialization parameter LOG\_ARCHIVE\_MAX\_PROCESSES. By default, this initialization parameter has a value of 2 and is rarely changed.
* **Checkpoint (CKPT).** Updates datafile headers whenever a checkpoint is performed.
* **Recover (RECO).** Automatically cleans up failed or suspended distributed transactions.
* **Dispatcher.** Optional background processes used when shared server configurations are deployed.
* **Global Cache Service (LMS).** Manages resources for Real Application Clusters and inter-instance resource control.
* **Job Queue.** Provides a scheduler service used to schedule user PL/SQL statements or procedures in batch.
* **Queue Monitor (QMNn).** Monitors Oracle Streams message queues with up to 10 monitoring processes supported.
* **Automatic Storage Management (ASM) processes.** RBAL coordinates rebalancing of activities for disk groups. ORBn performs the actual rebalancing. ASMB provides communication between the database and the ASM instance.

## Connecting to Oracle

To be able to better understand other memory structures (as PGA and SGA) we need to take a look at processes of connection to Oracle Database. We’ll take a look at the mechanics behind the two most common ways to have requests serviced by an Oracle server: **dedicated** server and **shared** server connections. We’ll take a brief look at how to establish TCP/IP connections; TCP/IP is the primary networking protocol used to connect over the network to Oracle. And we’ll look at how the listener process on our server, which is responsible for establishing the physical connection to the server, works differently in the cases of dedicated and shared server connections.

### Server Processes and Clients

To access a database, a user connects to the instance that provides access to the desired database. A program that accesses a database is really composed of two distinct pieces —a client program and a server process—that connect to the Oracle instance. For example, running the Oracle character-mode utility SQL\*Plus involves two processes:

* The SQL\*Plus process itself, acting as the client
* The Oracle server process, sometimes referred to as a shadow process, that provides the connection to the Oracle instance

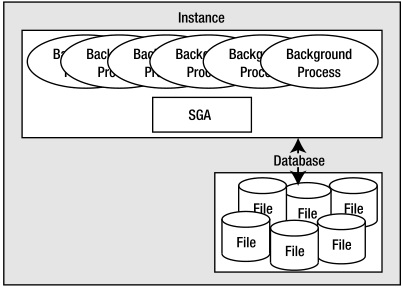
### Server process

The Oracle server process always runs on the computer on which the instance is running. The server process attaches to the shared memory used for the SGA and can read from it and write to it. As the name implies, the server process works for the client process—it reads and passes back the requested data, accepts and makes changes on behalf of the client, and so on. For example, when a client wants to read a row of data stored in a particular database block, the server process identifies the desired block and either retrieves it from the database buffer cache or reads it from the correct datafile and loads it into the database buffer cache. Then, if the user requests changes, the server process modifies the block in the cache and generates and stores the necessary redo information in the redo log buffer in the SGA. The server process, however, does not write the redo information from the log buffer to the redo logfiles, and it does not write the modified database block from the buffer cache to the datafile. These actions are performed by the Log Writer (LGWR) and Database Writer (DBWR) processes, respectively.

### Client process

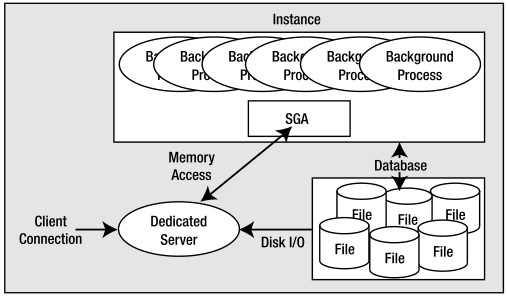
The client process can run on the same machine as the instance or on a separate computer. A network connects the two computers and provides a way for the two processes to talk to each other. In either case, the concept is essentially the same—two processes are involved in the interaction between a client and the database. When both processes are on the same machine, Oracle uses local communications via Inter Process Communication (IPC); when the client is on one machine and the database server is on another, Oracle uses Oracle Net over the network to communicate between the two machines.

### Dedicated Server



**Figure 3 Oracle Instance and Database**

Figure 3 and the **pstat** output presented a picture of what Oracle looks like immediately after starting. If we were now to log into this database using a dedicated server, we would see a new thread get created just to service us.



**Figure 4 Dedicated Server Configuration**

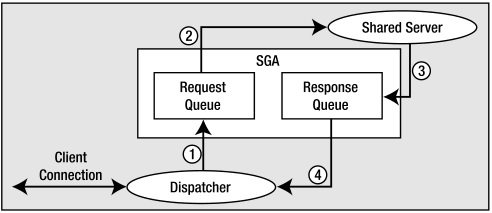
As noted, typically Oracle will create a new process for me when user log in. This is commonly referred to as the dedicated server configuration, since a server process will be dedicated to user for the life of my session. For each session, a new dedicated server will appear in a one-to-one mapping.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

### Shared Server

Oracle can also accept connections in a manner called shared server, in which you wouldn’t see an additional thread created or a new UNIX process appear for each user connection.

In shared server, Oracle uses a pool of shared processes for a large community of users. Shared servers are simply a connection pooling mechanism. Instead of having 10,000 dedicated servers (that’s a lot of processes or threads) for 10,000 database sessions, shared server lets us have a small percentage of this number of processes or threads, which would be (as the name implies) shared by all sessions. This allows Oracle to connect many more users to the instance than would otherwise be possible. Our machine might crumble under the load of managing 10,000 processes, but managing 100 or 1,000processes is doable. In shared server mode, the shared processes are generally started up with the database and appear in the ps list.

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



**Figure 5 Steps in a shared server request**

## Mechanics of Connecting over TCP/IP

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

$sqlplusscott/tiger@orcl

SQL\*Plus: Release 11.2.0.1.0 Production on Fri Dec 11 16:00:31 2009

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:

Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production

With the Partitioning, OLAP, Data Mining and Real Application Testing options

scott%ORA11GR2>

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

$ cat $ORACLE\_HOME/network/admin/tnsnames.ora

ORCL =

(DESCRIPTION =

(ADDRESS =

(PROTOCOL = TCP)

(HOST = somehost.somewhere.com)

(PORT = 1521)

)

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SERVICE\_NAME = orcl)

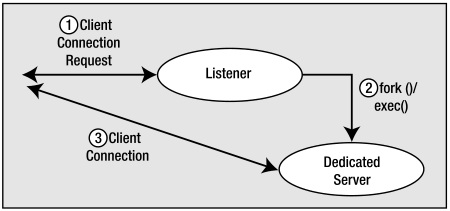
)

)

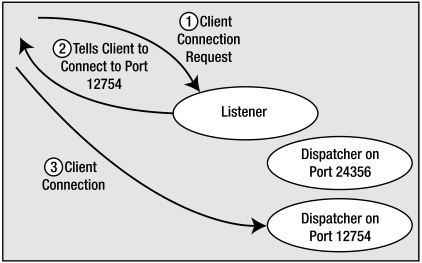
This configuration information allows the Oracle client software to map the **TNS** connection string we used, **orcl**, into something useful—namely, a hostname, a port on that host on which a listener process will accept connections, the service name of the database on the host to which we wish to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

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

Now that the client software knows where to connect to, it will open a TCP/IP socket connection to the server with the hostname somehost.somewhere.com on port **1521**. If the DBA for our server has installed and configured Oracle Net, and has the listener listening on port **1521** for connection requests, this connection may be accepted. In a network environment, we will be running a process called the TNS listener on our server. This listener process is what will get us physically connected to our database. When it receives the inbound connection request, it inspects the request and, using its own configuration files.

****

**Figure 6 Listner Process and Dedicated Server**



**Figure 7 Listener Process and Shared Server**

## The Process Global Area and User Global Area

The PGA is a process-specific piece of memory. In other words, it is memory specific to a single operating system process or thread. The PGA consists of session memory and a private SQL area. This memory is not accessible by any other process or thread in the system. It may grow (or even shrink) at runtime. The PGA is never allocated in Oracle's SGA; it is always allocated locally by the process or thread—the P in PGA stands for process or program; it is not shared.

The UGA is, in effect, your session's state. It is memory that your session must always be able to get to. The location of the UGA is wholly dependent on how you connect to Oracle. If you connect via a shared server, the UGA must be stored in a memory structure that every shared server process has access to—and that’s the SGA. In this way, your session can use any one of the shared servers, since any of them can read and write your session's data. On the other hand, if you are using a dedicated server connection, there’s no need for universal access to your session state, and the UGA becomes virtually synonymous with the PGA; it will, in fact, be contained in the PGA of your dedicated server. When you look at the system statistics, you'll find the UGA reported in the PGA in dedicated server mode (the PGA will be greater than or equal to the UGA memory used; the PGA memory size will include the UGA size as well).

So, the PGA contains process memory and may include the UGA. The other areas of PGA memory are generally used for in-memory sorting, bitmap merging, and hashing. It would be safe to say that, besides the UGA memory, these are the largest contributors by far to the PGA.

# SQL Execution

The following picture shows which processes and memory areas take participation in the process of execution of SQL:

****

**Figure 8 Oracle database**

# Source Books and Articles

1. Kyte T. Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions, Second Edition. Apress, 2010.
2. Morton K., & Osborne K., & Sands R., & Shamsudeen R., & Still J. Pro Oracle SQL. Apress, 2013.
3. Greenwald R., & Stackowiak R., & Stern J. Oracle Essentials, Fifth Edition. Sebastopol: O’Reilly Media, Inc., 2013.