Oracle® Database

Concepts

11*g* Release 2 (11.2)

E40540-03

April 2015

ORACLE®

Oracle Database Concepts, 11*g* Release 2 (11.2)

E40540-03

Primary Authors: Lance Ashdown, Tom Kyte

Contributors: Drew Adams, David Austin, Vladimir Barriere, Hermann Baer, David Brower, Jonathan Creighton, Bjørn Engsig, Steve Fogel, Bill Habeck, Bill Hodak, Yong Hu, Pat Huey, Vikram Kapoor, Feroz Khan, Jonathan Klein, Sachin Kulkarni, Paul Lane, Adam Lee, Yunrui Li, Bryn Llewellyn, Rich Long, Barb Lundhild, Neil Macnaughton, Vineet Marwah, Mughees Minhas, Sheila Moore, Valarie Moore, Gopal Mulagund, Paul Needham, Gregory Pongracz, John Russell, Vivian Schupmann, Shrikanth Shankar, Cathy Shea, Susan Shepard, Jim Stenoish, Juan Tellez, Lawrence To, Randy Urbano, Badhri Varanasi, Simon Watt, Steve Wertheimer, Daniel Wong

13

Oracle Database Instance

This chapter explains the nature of an Oracle database instance, the parameter and diagnostic files associated with an instance, and what occurs during instance creation and the opening and closing of a database.

This chapter contains the following sections:

Introduction to the Oracle Database Instance

A database **instance** is a set of memory structures that manage database files. A **database** is a set of physical files on disk created by the `CREATE DATABASE` statement. The instance manages its associated data and serves the users of the database.

Every running Oracle database is associated with at least one Oracle database instance. Because an instance exists in memory and a database exists on disk, an instance can exist without a database and a database can exist without an instance.

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:

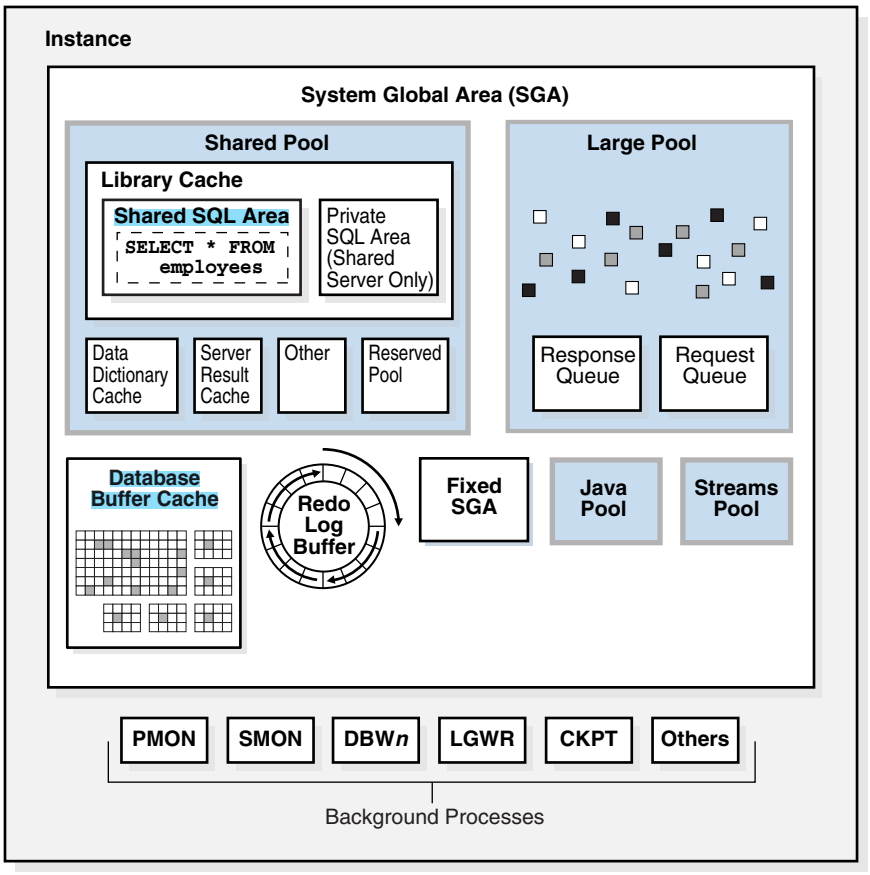
- Maintaining internal data structures that are accessed by many processes and threads concurrently
- Caching data blocks read from disk
- Buffering redo data before writing it to the online redo log files
- Storing SQL **execution plans**

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 varies according to operating system.

A database instance includes background processes. Server processes, and the process memory allocated in these processes, also exist in the instance. The instance continues to function when server processes terminate.

Figure 13–1 shows the main components of an Oracle database instance.

Figure 13–1 Database Instance



See Also:

- "Overview of the System Global Area" on page 14-8
- "Overview of Background Processes" on page 15-7

Database Instance Configurations

You can run Oracle Database in either of the following mutually exclusive configurations:

- Single-instance configuration

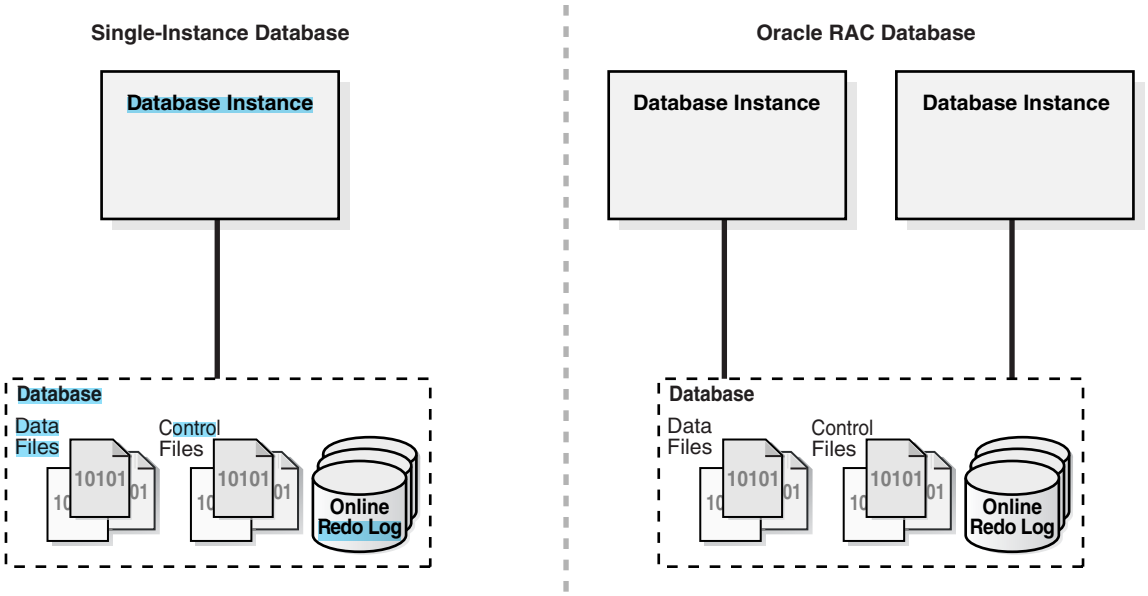
  A one-to-one relationship exists between the database and an instance.

- Oracle Real Application Clusters (Oracle RAC) configuration

  A one-to-many relationship exists between the database and instances.

Figure 13–2 shows possible database instance configurations.

Figure 13–2 Database Instance Configurations



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.

Note: This chapter discusses a single-instance database configuration unless otherwise noted.

Multiple instances can run concurrently on the same computer, each accessing its own database. For example, a computer can host two distinct databases: `prod1` and `prod2`. One database instance manages `prod1`, while a separate instance manages `prod2`.

> **See Also:** *Oracle Real Application Clusters Administration and Deployment Guide* for information specific to Oracle RAC

Duration of an Instance

An instance begins when it is created with the `STARTUP` command and ends when it is terminated. During this period, an instance can associate itself with one and only one database. Furthermore, the instance can mount a database only once, close it only once, and open it only once. After a database has been closed or shut down, you must start a *different* instance to mount and open this database.

Table 13–1 illustrates a database instance attempting to reopen a database that it previously closed.

Table 13–1 Duration of an Instance

Statement	Explanation
SQL> STARTUP ORACLE instance started. Total System Global Area 468729856 bytes Fixed Size 1333556 bytes Variable Size 440403660 bytes Database Buffers 16777216 bytes Redo Buffers 10215424 bytes Database mounted. Database opened.	The STARTUP command creates an instance, which mounts and opens the database.
SQL> SELECT TO_CHAR(STARTUP_TIME,'MON-DD-RR HH24:MI:SS') AS "Inst Start Time" FROM V$INSTANCE; Inst Start Time ------------------ JUN-18-11 **13:14:48**	This query shows the time that the current instance was started.
SQL> SHUTDOWN IMMEDIATE	The instance closes the database and shuts down, ending the life of this instance.
SQL> STARTUP Oracle instance started. . . .	The STARTUP command creates a new instance and mounts and open the database.
SQL> SELECT TO_CHAR(STARTUP_TIME,'MON-DD-RR HH24:MI:SS') AS "Inst Start Time" FROM V$INSTANCE; Inst Start Time ------------------ JUN-18-11 **13:16:40**	This query shows the time that the current instance was started. The different start time shows that this instance is different from the one that shut down the database.

Oracle System Identifier (SID)

The **system identifier (SID)** is a unique name for an Oracle database instance on a specific host. On UNIX and Linux, Oracle Database uses the SID and **Oracle home** values to create a key to shared memory. Also, the SID is used by default to locate the parameter file, which is used to locate relevant files such as the database control files.

On most platforms, the ORACLE_SID environment variable sets the SID, whereas the ORACLE_HOME variable sets the Oracle home. When connecting to an instance, clients can specify the SID in an Oracle Net connection or use a net service name. Oracle Database converts a service name into an ORACLE_HOME and ORACLE_SID.

See Also:

- "Service Names" on page 16-8
- *Oracle Database Administrator's Guide* to learn how to specify an Oracle SID

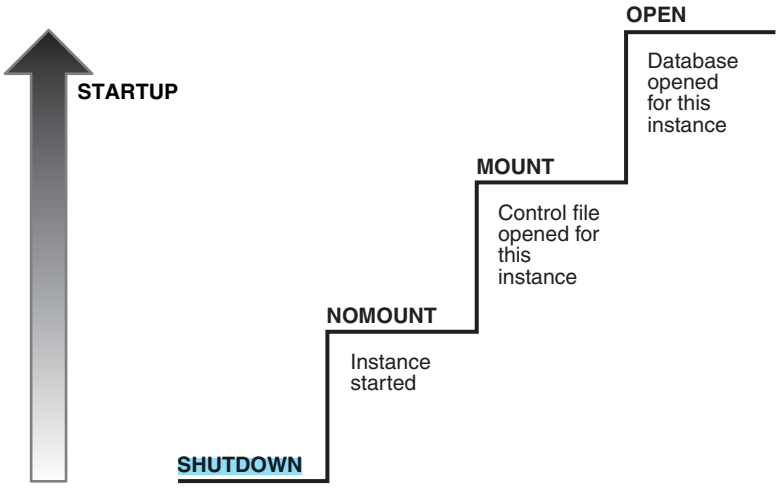
Overview of Instance Startup and Shutdown

A database instance provides user access to a database. This section explains the possible states of the instance and the database.

Overview of Instance and Database Startup

In a typical use case, you manually start an instance and then mount and open the database, making it available for users. You can use the SQL*Plus `STARTUP` command, Oracle Enterprise Manager (Enterprise Manager), or the SRVCTL utility to perform these steps. Figure 13–3 shows how a database progresses from a shutdown state to an open state.

Figure 13–3 Instance and Database Startup Sequence



A database goes through the following phases when it proceeds from a shutdown state to an open database state:

1. Instance started without mounting database

   The instance is started, but is not yet associated with a database.

   "How an Instance Is Started" on page 13-6 explains this stage.

2. Database mounted

   The instance is started and is associated with a database by reading its **control file** (see "Overview of Control Files" on page 11-10). The database is closed to users.

   "How a Database Is Mounted" on page 13-6 explains this stage.

3. Database open

   The instance is started and is associated with an open database. The data contained in the **data files** is accessible to authorized users.

   "How a Database Is Opened" on page 13-7 explains this stage.

See Also:

- "Oracle Enterprise Manager" on page 18-2
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to start an instance
- *Oracle Database Administrator's Guide* to learn how to use SRVCTL

Connection with Administrator Privileges

Database startup and shutdown are powerful administrative options that are restricted to users who connect to Oracle Database with administrator privileges. Normal users do not have control over the current status of an Oracle database.

Depending on the operating system, one of the following conditions establishes administrator privileges for a user:

- The operating system privileges of the user enable him or her to connect using administrator privileges.
- The user is granted the `SYSDBA` or `SYSOPER` system privileges and the database uses password files to authenticate database administrators over the network.

`SYSDBA` and `SYSOPER` are special **system privileges** that enable access to a database instance even when the database is not open. Control of these privileges is outside of the database itself.

When you connect with the `SYSDBA` system privilege, you are in the schema owned by `SYS`. When you connect as `SYSOPER`, you are in the public schema. `SYSOPER` privileges are a subset of `SYSDBA` privileges.

> **See Also:**
>
> - "SYS and SYSTEM Schemas" on page 2-5
> - "Overview of Database Security" on page 17-1 to learn about password files and authentication for database administrators
> - *Oracle Database Administrator's Guide* to learn about `SYSDBA` and `SYSOPER`

How an Instance Is Started

When Oracle Database starts an instance, it performs the following basic steps:

1. Searches for a **server parameter file** in a platform-specific default location and, if not found, for a text **initialization parameter file** (specifying `STARTUP` with the `SPFILE` or `PFILE` parameters overrides the default behavior)
2. Reads the parameter file to determine the values of initialization parameters
3. Allocates the SGA based on the initialization parameter settings
4. Starts the Oracle background processes
5. Opens the **alert log** and trace files and writes all explicit parameter settings to the **alert log** in valid parameter syntax

At this stage, no database is associated with the instance. Scenarios that require a `NOMOUNT` state include database creation and certain backup and recovery operations.

> **See Also:** *Oracle Database Administrator's Guide* to learn how to manage initialization parameters using a server parameter file

How a Database Is Mounted

The instance **mounts** a database to associate the database with this instance. To mount the database, the instance obtains the names of the database control files specified in the `CONTROL_FILES` initialization parameter and opens the files. Oracle Database reads the control files to find the names of the data files and the online redo log files that it will attempt to access when opening the database.

In a **mounted database**, the database is closed and accessible only to database administrators. Administrators can keep the database closed while completing specific maintenance operations. However, the database is not available for normal operations.

If Oracle Database allows multiple instances to mount the same database concurrently, then the `CLUSTER_DATABASE` initialization parameter setting can make the database available to multiple instances. Database behavior depends on the setting:

- If `CLUSTER_DATABASE` is `false` (default) for the first instance that mounts a database, then only this instance can mount the database.
- If `CLUSTER_DATABASE` is `true` for the first instance, then other instances can mount the database if their `CLUSTER_DATABASE` parameter settings are set to `true`. The number of instances that can mount the database is subject to a predetermined maximum specified when creating the database.

See Also:

- *Oracle Database Administrator's Guide* to learn how to mount a database
- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about the use of multiple instances with a single database

How a Database Is Opened

Opening a mounted database makes it available for normal database operations. Any valid user can connect to an open database and access its information. Usually, a database administrator opens the database to make it available for general use.

When you open the database, Oracle Database performs the following actions:

- Opens the online data files in tablespaces other than undo tablespaces

  If a tablespace was offline when the database was previously shut down (see "Online and Offline Tablespaces" on page 12-35), then the tablespace and its corresponding data files will be offline when the database reopens.

- Acquires an undo tablespace

  If multiple undo tablespaces exists, then the `UNDO_TABLESPACE` initialization parameter designates the undo tablespace to use. If this parameter is not set, then the first available undo tablespace is chosen.

- Opens the online redo log files

See Also: "Data Repair" on page 18-12

Read-Only Mode By default, the database opens in **read/write mode**. In this mode, users can make changes to the data, generating redo in the online redo log. Alternatively, you can open in **read-only mode** to prevent data modification by user transactions.

Note: By default, a physical **standby database** opens in read-only mode. See *Oracle Data Guard Concepts and Administration*.

Read-only mode restricts database access to read-only transactions, which cannot write to data files or to online redo log files. However, the database can perform recovery or operations that change the database state without generating redo. For example, in read-only mode:

- Data files can be taken offline and online. However, you cannot take permanent tablespaces offline.
- Offline data files and tablespaces can be recovered.
- The control file remains available for updates about the state of the database.
- Temporary tablespaces created with the `CREATE TEMPORARY TABLESPACE` statement are read/write.
- Writes to operating system audit trails, trace files, and alert logs can continue.

> **See Also:** *Oracle Database Administrator's Guide* to learn how to open a database in read-only mode

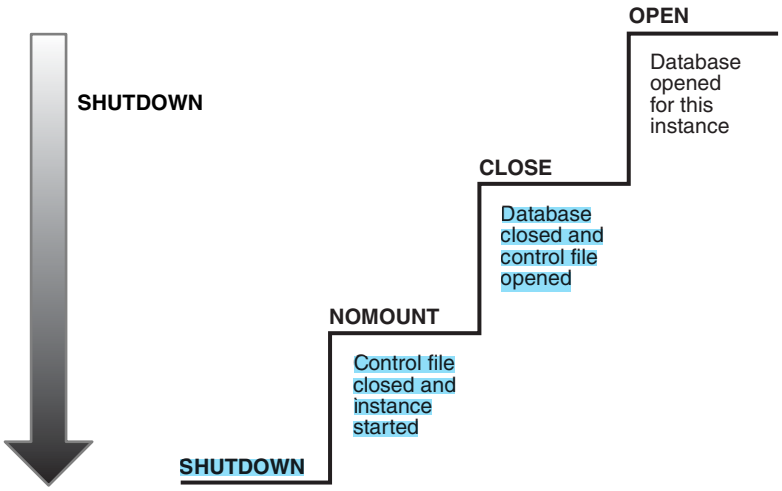
Database File Checks If any of the data files or redo log files are not present when the instance attempts to open the database, or if the files are present but fail consistency tests, then the database returns an error. Media recovery may be required.

> **See Also:** "Backup and Recovery" on page 18-9

Overview of Database and Instance Shutdown

In a typical use case, you manually shut down the database, making it unavailable for users while you perform maintenance or other administrative tasks. You can use the SQL*Plus `SHUTDOWN` command or Enterprise Manager to perform these steps. Figure 13–4 shows the progression from an open state to a consistent shutdown.

Figure 13–4 Instance and Database Shutdown Sequence



Oracle Database automatically performs the following steps whenever an open database is shut down consistently:

1. Database closed

   The database is mounted, but online data files and redo log files are closed.

   "How a Database Is Closed" explains this stage.

2. Database unmounted

   The instance is started, but is no longer associated with the control file of the database.

"How a Database Is Unmounted" explains this stage.

3. Database instance shut down

   The database instance is no longer started.

   "How an Instance Is Shut Down" explains this stage.

Oracle Database does not go through all of the preceding steps in an instance failure or `SHUTDOWN ABORT`, which immediately terminates the instance.

> **See Also:** *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to shut down a database

Shutdown Modes

A database administrator with `SYSDBA` or `SYSOPER` privileges can shut down the database using the SQL*Plus `SHUTDOWN` command or Enterprise Manager. The `SHUTDOWN` command has options that determine shutdown behavior. Table 13–2 summarizes the behavior of the different shutdown modes.

Table 13–2 Shutdown Modes

Database Behavior	ABORT	IMMEDIATE	TRANSACTIONAL	NORMAL
Permits new user connections	No	No	No	No
Waits until current sessions end	No	No	No	Yes
Waits until current transactions end	No	No	Yes	Yes
Performs a **checkpoint** and closes open files	No	Yes	Yes	Yes

The possible `SHUTDOWN` statements are:

- `SHUTDOWN ABORT`

  This mode is intended for emergency situations, such as when no other form of shutdown is successful. This mode of shutdown is the fastest. However, a subsequent open of this database may take substantially longer because instance recovery must be performed to make the data files consistent.

  > **Note:** Because `SHUTDOWN ABORT` does not checkpoint the open data files, instance recovery is necessary before the database can reopen. The other shutdown modes do not require instance recovery before the database can reopen.

- `SHUTDOWN IMMEDIATE`

  This mode is typically the fastest next to `SHUTDOWN ABORT`. Oracle Database terminates any executing SQL statements and disconnects users. Active transactions are terminated and uncommitted changes are rolled back.

- `SHUTDOWN TRANSACTIONAL`

  This mode prevents users from starting new transactions, but waits for all current transactions to complete before shutting down. This mode can take a significant amount of time depending on the nature of the current transactions.

- `SHUTDOWN NORMAL`

  This is the default mode of shutdown. The database waits for all connected users to disconnect before shutting down.

  See Also:

  - *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn about the different shutdown modes
  - *SQL*Plus User's Guide and Reference* to learn about the `SHUTDOWN` command

How a Database Is Closed

The database close operation is implicit in a database shutdown. The nature of the operation depends on whether the database shutdown is normal or abnormal.

How a Database Is Closed During Normal Shutdown When a database is closed as part of a `SHUTDOWN` with any option other than `ABORT`, Oracle Database writes data in the SGA to the data files and online redo log files. Next, the database closes online data files and online redo log files. Any offline data files of offline tablespaces have been closed already. When the database reopens, any tablespace that was offline remains offline.

At this stage, the database is closed and inaccessible for normal operations. The control files remain open after a database is closed.

How a Database Is Closed During Abnormal Shutdown If a `SHUTDOWN ABORT` or abnormal termination occurs, then the instance of an open database closes and shuts down the database instantaneously. Oracle Database does not write data in the buffers of the SGA to the data files and redo log files. The subsequent reopening of the database requires instance recovery, which Oracle Database performs automatically.

How a Database Is Unmounted

After the database is closed, Oracle Database unmounts the database to disassociate it from the instance. After a database is unmounted, Oracle Database closes the control files of the database. At this point, the instance remains in memory.

How an Instance Is Shut Down

The final step in database shutdown is shutting down the instance. When the database instance is shut down, the SGA is removed from memory and the background processes are terminated.

In unusual circumstances, shutdown of an instance may not occur cleanly. Memory structures may not be removed from memory or one of the background processes may not be terminated. When remnants of a previous instance exist, a subsequent instance startup may fail. In such situations, you can force the new instance to start by removing the remnants of the previous instance and then starting a new instance, or by issuing a `SHUTDOWN ABORT` statement in SQL*Plus or using Enterprise Manager.

See Also: *Oracle Database Administrator's Guide* for more detailed information about database shutdown

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:

- 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.

- The writing of modified database buffers in the **database buffer cache** to disk

See Also: "System Change Numbers (SCNs)" on page 10-5

Purpose of Checkpoints

Oracle Database uses checkpoints to achieve the following goals:

- Reduce the time required for recovery in case of an instance or media failure
- Ensure that dirty buffers in the buffer cache are written to disk regularly
- Ensure that all committed data is written to disk during a consistent shutdown

When Oracle Database Initiates Checkpoints

The **checkpoint process (CKPT)** is responsible for writing checkpoints to the data file headers and control file. Checkpoints occur in a variety of situations. For example, Oracle Database uses the following types of checkpoints:

- Thread checkpoints

  The database writes to disk all buffers modified by redo in a specific thread before a certain target. The set of thread checkpoints on all instances in a database is a **database checkpoint**. Thread checkpoints occur in the following situations:

  - Consistent database shutdown
  - `ALTER SYSTEM CHECKPOINT` statement
  - Online redo log switch
  - `ALTER DATABASE BEGIN BACKUP` statement

- Tablespace and data file checkpoints

  The database writes to disk all buffers modified by redo before a specific target. A tablespace checkpoint is a set of data file checkpoints, one for each data file in the tablespace. These checkpoints occur in a variety of situations, including making a tablespace read-only or taking it offline normal, shrinking a data file, or executing `ALTER TABLESPACE BEGIN BACKUP`.

- Incremental checkpoints

  An incremental checkpoint is a type of thread checkpoint partly intended to avoid writing large numbers of blocks at online redo log switches. DBW*n* checks at least every three seconds to determine whether it has work to do. When DBW*n* writes dirty buffers, it advances the checkpoint position, causing CKPT to write the checkpoint position to the control file, but not to the data file headers.

Other types of checkpoints include instance and media recovery checkpoints and checkpoints when schema objects are dropped or truncated.

See Also:

- "Checkpoint Process (CKPT)" on page 15-10
- *Oracle Real Application Clusters Administration and Deployment Guide* for information about global checkpoints in Oracle RAC

Overview of Instance Recovery

Instance recovery is the process of applying records in the **online redo log** to data files to reconstruct changes made after the most recent **checkpoint**. Instance recovery occurs automatically when an administrator attempts to open a database that was previously shut down inconsistently.

Purpose of Instance Recovery

Instance recovery ensures that the database is in a consistent state after an instance failure. The files of a database can be left in an inconsistent state because of how Oracle Database manages database changes.

A **redo thread** is a record of all of the changes generated by an instance. A single-instance database has one thread of redo, whereas an Oracle RAC database has multiple redo threads, one for each database instance.

When a **transaction** is committed, **log writer (LGWR)** writes both the remaining redo entries in memory and the transaction SCN to the **online redo log**. However, the **database writer (DBW)** process writes modified data blocks to the data files whenever it is most efficient. For this reason, uncommitted changes may temporarily exist in the data files while committed changes do not yet exist in the data files.

If an instance of an open database fails, either because of a `SHUTDOWN ABORT` statement or abnormal termination, then the following situations can result:

- Data blocks committed by a transaction are not written to the data files and appear only in the **online redo log**. These changes must be reapplied to the database.
- The data files contains changes that had not been committed when the instance failed. These changes must be rolled back to ensure transactional consistency.

Instance recovery uses only online redo log files and current online data files to synchronize the data files and ensure that they are consistent.

See Also:

- "Database Writer Process (DBWn)" on page 15-8 and "Database Buffer Cache" on page 14-9
- "Introduction to Data Concurrency and Consistency" on page 9-1

When Oracle Database Performs Instance Recovery

Whether instance recovery is required depends on the state of the redo threads. A redo thread is marked open in the control file when a database instance opens in read/write mode, and is marked closed when the instance is shut down consistently. If redo threads are marked open in the control file, but no live instances hold the thread enqueues corresponding to these threads, then the database requires instance recovery.

Oracle Database performs instance recovery automatically in the following situations:

- The database opens for the first time after the failure of a single-instance database or all instances of an Oracle RAC database. This form of instance recovery is also called **crash recovery**. Oracle Database recovers the online redo threads of the terminated instances together.
- Some but not all instances of an Oracle RAC database fail. Instance recovery is performed automatically by a surviving instance in the configuration.

The SMON background process performs instance recovery, applying online redo automatically. No user intervention is required.

See Also:

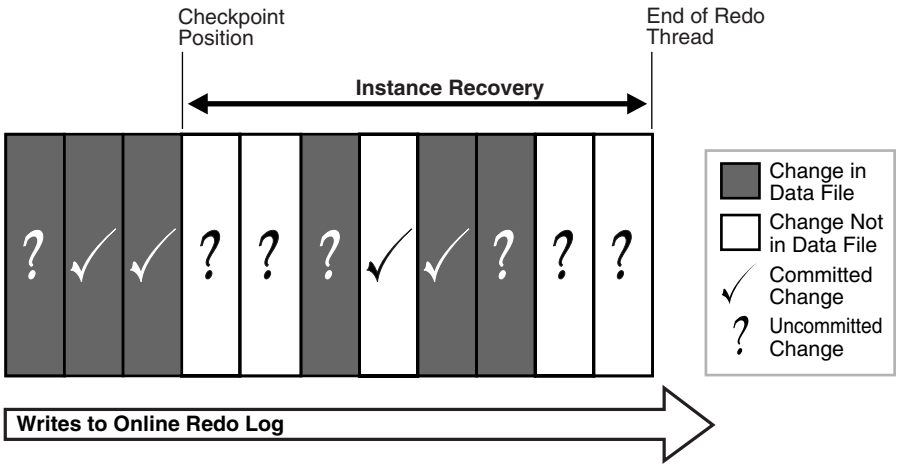
- "System Monitor Process (SMON)" on page 15-8
- *Oracle Real Application Clusters Administration and Deployment Guide* to learn about instance recovery in an Oracle RAC database

Importance of Checkpoints for Instance Recovery

Instance recovery uses checkpoints to determine which changes must be applied to the data files. The checkpoint position guarantees that every committed change with an SCN *lower than* the checkpoint SCN is saved to the data files.

Figure 13–5 depicts the redo thread in the online redo log.

Figure 13–5 Checkpoint Position in Online Redo Log



During instance recovery, the database must apply the changes that occur between the checkpoint position and the end of the redo thread. As shown in Figure 13–5, some changes may already have been written to the data files. However, only changes with SCNs lower than the checkpoint position are *guaranteed* to be on disk.

See Also: *Oracle Database Performance Tuning Guide* to learn how to limit instance recovery time

Instance Recovery Phases

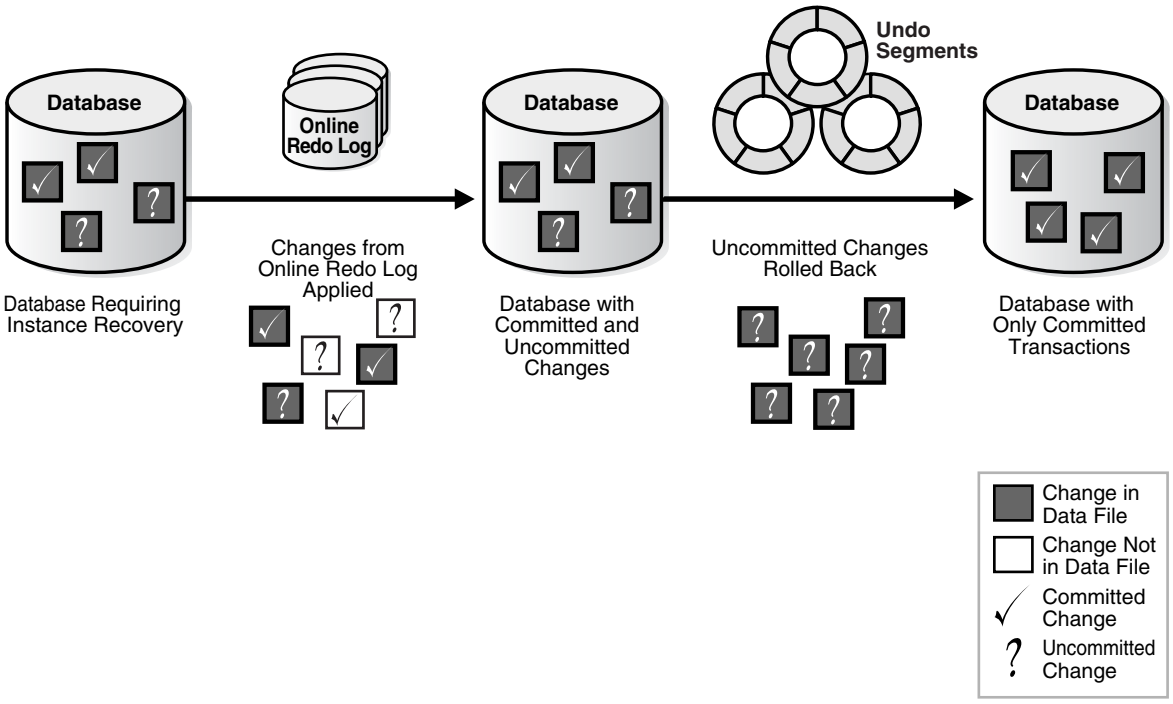
The first phase of instance recovery is called **cache recovery** or **rolling forward**, and involves reapplying all of the changes recorded in the online redo log to the data files. Because rollback data is recorded in the online redo log, rolling forward also regenerates the corresponding undo segments.

Rolling forward proceeds through as many online redo log files as necessary to bring the database forward in time. After rolling forward, the data blocks contain all committed changes recorded in the online redo log files. These files could also contain uncommitted changes that were either saved to the data files before the failure, or were recorded in the online redo log and introduced during cache recovery.

After the roll forward, any changes that were not committed must be undone. Oracle Database uses the checkpoint position, which guarantees that every committed change with an SCN lower than the checkpoint SCN is saved on disk. Oracle Database applies undo blocks to roll back uncommitted changes in data blocks that were written before the failure or introduced during cache recovery. This phase is called **rolling back** or **transaction recovery**.

Figure 13–6 illustrates rolling forward and rolling back, the two steps necessary to recover from database instance failure.

Figure 13–6 Basic Instance Recovery Steps: Rolling Forward and Rolling Back



Oracle Database can roll back multiple transactions simultaneously as needed. All transactions that were active at the time of failure are marked as terminated. Instead of waiting for the SMON process to roll back terminated transactions, new transactions can roll back individual blocks themselves to obtain the required data.

See Also:

- "Undo Segments" on page 12-24 to learn more about undo data
- *Oracle Database Performance Tuning Guide* for a discussion of instance recovery mechanics and tuning

Overview of Parameter Files

To start a database instance, Oracle Database must read either a **server parameter file**, which is recommended, or a **text initialization parameter file**, which is a legacy implementation. These files contain a list of configuration parameters.

To create a database manually, you must start an instance with a parameter file and then issue a `CREATE DATABASE` command. Thus, the instance and parameter file can exist even when the database itself does not exist.

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 adequate in most cases.

Functional Groups of Initialization Parameters

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

- Parameters that name entities such as files or directories
- Parameters that set limits for a process, database resource, or the database itself
- 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: basic and advanced. In most cases, 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 (see "Tools for Database Installation and Configuration" on page 18-4). 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.

See Also:

- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to specify initialization parameters
- *Oracle Database Reference* for an explanation of the types of initialization parameters
- *Oracle Database Reference* for a description of `V$PARAMETER` and *SQL*Plus User's Guide and Reference* for `SHOW PARAMETER` syntax

Server Parameter Files

A **server parameter file** is a repository for initialization parameters that is managed by Oracle Database. A server parameter file has the following key characteristics:

- Only one server parameter file exists for a database. This file must reside on the database host.
- The server parameter file is written to and read by only by Oracle Database, not by client applications.
- The server parameter file is binary and cannot be modified by a text editor.
- Initialization parameters stored in the server parameter file are persistent. Any changes made to the parameters while a database instance is running can persist across instance shutdown and startup.

A server parameter file eliminates the need to maintain multiple text initialization parameter files for client applications. A server parameter file is initially built from a text initialization parameter file using the `CREATE SPFILE` statement. It can also be created directly by the Database Configuration Assistant.

See Also:

- *Oracle Database Administrator's Guide* to learn more about server parameter files
- *Oracle Database SQL Language Reference* to learn about `CREATE SPFILE`

Text Initialization Parameter Files

A **text initialization parameter file** is a text file that contains a list of initialization parameters. This type of parameter file, which is a legacy implementation of the parameter file, has the following key characteristics:

- When starting up or shutting down a database, the text initialization parameter file must reside on the same host as the client application that connects to the database.
- A text initialization parameter file is text-based, not binary.
- Oracle Database can read but not write to the text initialization parameter file. To change the parameter values you must manually alter the file with a text editor.
- Changes to initialization parameter values by `ALTER SYSTEM` are only in effect for the current instance. You must manually update the text initialization parameter file and restart the instance for the changes to be known.

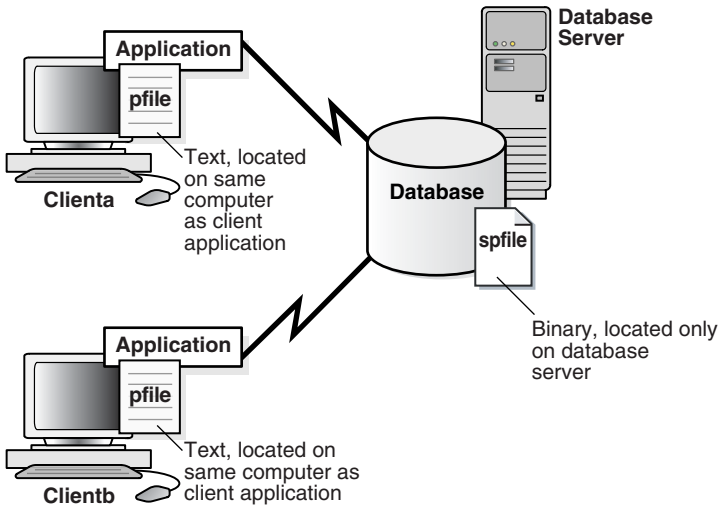
The text initialization parameter file contains a series of `key=value` pairs, one per line. For example, a portion of an initialization parameter file could look as follows:

```
db_name=sample
control_files=/disk1/oradata/sample_cf.dbf
db_block_size=8192
open_cursors=52
undo_management=auto
shared_pool_size=280M
pga_aggregate_target=29M
.
.
.
```

To illustrate the manageability problems that text parameter files can create, assume that you use computers `clienta` and `clientb` and must be able to start the database with SQL*Plus on either computer. In this case, two separate text initialization parameter files must exist, one on each computer, as shown in Figure 13–7. A server

parameter file solves the problem of the proliferation of parameter files.

Figure 13–7 Multiple Initialization Parameter Files



See Also:

- *Oracle Database Administrator's Guide* to learn more about text initialization parameter files
- *Oracle Database SQL Language Reference* to learn about `CREATE PFILE`

Modification of Initialization Parameter Values

You can adjust initialization parameters to modify the behavior of a database. The classification of parameters as **static** or **dynamic** determines how they can be modified. Table 13–3 summarizes the differences.

Table 13–3 Static and Dynamic Initialization Parameters

Characteristic	Static	Dynamic
Requires modification of the parameter file (text or server)	Yes	No
Requires database instance restart before setting takes affect	Yes	No
Described as "Modifiable" in *Oracle Database Reference* initialization parameter entry	No	Yes
Modifiable only for the database or instance	Yes	No

Static parameters include `DB_BLOCK_SIZE`, `DB_NAME`, and `COMPATIBLE`. Dynamic parameters are grouped into **session-level parameters**, which affect only the current user session, and **system-level parameters**, which affect the database and all sessions. For example, `MEMORY_TARGET` is a system-level parameter, while `NLS_DATE_FORMAT` is a session-level parameter (see "Locale-Specific Settings" on page 19-10).

The **scope** of a parameter change depends on when the change takes effect. When an instance has been started with a server parameter file, you can use the `ALTER SYSTEM SET` statement to change values for system-level parameters as follows:

- `SCOPE=MEMORY`

Changes apply to the database instance only. The change will not persist if the database is shut down and restarted.

- `SCOPE=SPFILE`

  Changes are written to the server parameter file but do not affect the current instance. Thus, the changes do not take effect until the instance is restarted.

  Note: You must specify `SPFILE` when changing the value of a parameter described as not modifiable in *Oracle Database Reference*.

- `SCOPE=BOTH`

  Changes are written both to memory and to the server parameter file. This is the default scope when the database is using a server parameter file.

The database prints the new value and the old value of an initialization parameter to the alert log. As a preventative measure, the database validates changes of basic parameter to prevent illegal values from being written to the server parameter file.

See Also:

- *Oracle Database Administrator's Guide* to learn how to change initialization parameter settings
- *Oracle Database Reference* for descriptions of all initialization parameters
- *Oracle Database SQL Language Reference* for `ALTER SYSTEM` syntax and semantics

Overview of Diagnostic Files

Oracle Database includes a **fault diagnosability infrastructure** for preventing, detecting, diagnosing, and resolving database problems. Problems include critical errors such as code bugs, metadata corruption, and customer data corruption.

The goals of the advanced fault diagnosability infrastructure are the following:

- Detecting problems proactively
- Limiting damage and interruptions after a problem is detected
- Reducing problem diagnostic and resolution time
- Simplifying customer interaction with Oracle Support

Automatic Diagnostic Repository

Automatic Diagnostic Repository (ADR) is a file-based repository that stores database diagnostic data such as trace files, the alert log, and Health Monitor reports. Key characteristics of ADR include:

- Unified directory structure
- Consistent diagnostic data formats
- Unified tool set

The preceding characteristics enable customers and Oracle Support to correlate and analyze diagnostic data across multiple Oracle instances, components, and products.

ADR is located *outside* the database, which enables Oracle Database to access and manage ADR when the physical database is unavailable. An instance can create ADR before a database has been created.

Problems and Incidents

ADR proactively tracks **problems**, which are critical errors in the database. Critical errors manifest as internal errors, such as `ORA-600`, or other severe errors. Each problem has a **problem key**, which is a text string that describes the problem.

When a problem occurs multiple times, ADR creates a time-stamped **incident** for each occurrence. An incident is uniquely identified by a numeric **incident ID**. When an incident occurs, ADR sends an **incident alert** to Enterprise Manager. Diagnosis and resolution of a critical error usually starts with an incident alert.

Because a problem could generate many incidents in a short time, ADR applies flood control to incident generation after certain thresholds are reached. A **flood-controlled incident** generates an alert log entry, but does not generate incident dumps. In this way, ADR informs you that a critical error is ongoing without overloading the system with diagnostic data.

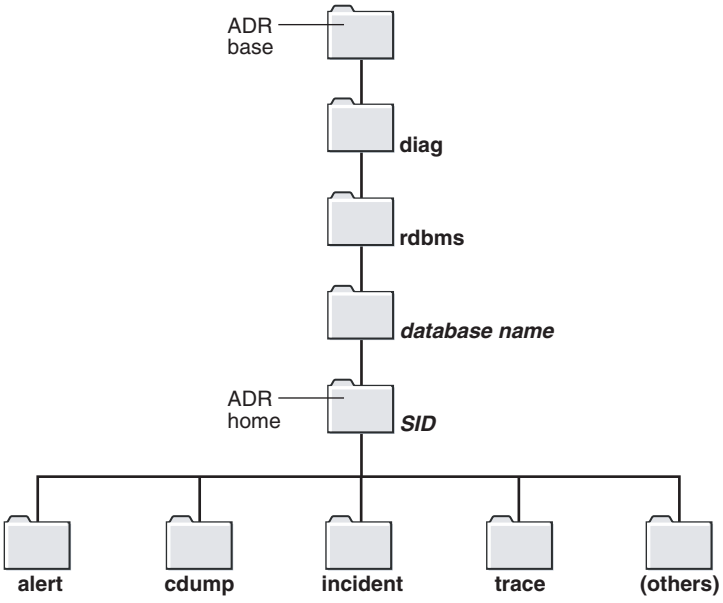
> **See Also:** *Oracle Database Administrator's Guide* for detailed information about the fault diagnosability infrastructure

ADR Structure

The **ADR base** is the ADR root directory. The ADR base can contain multiple ADR homes, where each **ADR home** is the root directory for all diagnostic data—traces, dumps, the alert log, and so on—for an instance of an Oracle product or component. For example, in an Oracle RAC environment with shared storage and ASM, each database instance and each ASM instance has its own ADR home.

Figure 13–8 illustrates the ADR directory hierarchy for a database instance. Other ADR homes for other Oracle products or components, such as ASM or Oracle Net Services, can exist within this hierarchy, under the same ADR base.

Figure 13–8 ADR Directory Structure for an Oracle Database Instance



As the following Linux example shows, when you start an instance with a unique SID and database name *before* creating a database, Oracle Database creates ADR by default as a directory structure in the host file system. The SID and database name form part of the path name for files in the ADR Home.

Example 13–1 Creation of ADR

```
% setenv ORACLE_SID osi
% echo "DB_NAME=dbn" > init.ora
% sqlplus / as sysdba
.
.
.
Connected to an idle instance.

SQL> STARTUP NOMOUNT PFILE="./init.ora"
ORACLE instance started.

Total System Global Area  146472960 bytes
Fixed Size                  1317424 bytes
Variable Size              92276176 bytes
Database Buffers           50331648 bytes
Redo Buffers                2547712 bytes

SQL> SELECT NAME, VALUE FROM V$DIAG_INFO;

NAME                 VALUE
-------------------- --------------------------------------------------
Diag Enabled         TRUE
ADR Base             /u01/oracle/log
ADR Home             /u01/oracle/log/diag/rdbms/dbn/osi
Diag Trace           /u01/oracle/log/diag/rdbms/dbn/osi/trace
Diag Alert           /u01/oracle/log/diag/rdbms/dbn/osi/alert
Diag Incident        /u01/oracle/log/diag/rdbms/dbn/osi/incident
Diag Cdump           /u01/oracle/log/diag/rdbms/dbn/osi/cdump
Health Monitor       /u01/oracle/log/diag/rdbms/dbn/osi/hm
Default Trace File   /u01/oracle/log/diag/rdbms/dbn/osi/trace/osi_ora_10533.trc
Active Problem Count 0
Active Incident Count 0
```

The following sections describe the contents of ADR.

Alert Log

Each database has an **alert log**, which is an XML file containing a chronological log of database messages and errors. The alert log contents include the following:

- All internal errors (`ORA-600`), block corruption errors (`ORA-1578`), and **deadlock** errors (`ORA-60`)
- Administrative operations such as **DDL** statements and the SQL*Plus commands `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`
- Several messages and errors relating to the functions of shared server and dispatcher processes
- Errors during the automatic refresh of a materialized view

Oracle Database uses the alert log as an alternative to displaying information in the Enterprise Manager GUI. If an administrative operation is successful, then Oracle Database writes a message to the alert log as "completed" along with a time stamp.

Oracle Database creates an alert log in the `alert` subdirectory shown in Figure 13–8 when you first start a database instance, even if no database has been created yet. The following example shows a portion of a text-only alert log:

```
Fri Jun 19 17:05:34 2011
Starting ORACLE instance (normal)
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Shared memory segment for instance monitoring created
Picked latch-free SCN scheme 2
Autotune of undo retention is turned on.
IMODE=BR
ILAT =12
LICENSE_MAX_USERS = 0
SYS auditing is disabled
Starting up ORACLE RDBMS Version: 11.2.0.0.0.
Using parameter settings in client-side pfile
.
.
.
System parameters with nondefault values:
  db_name                  = "my_test"
Fri Jun 19 17:05:37 2011
PMON started with pid=2, OS id=10329
Fri Jun 19 17:05:37 2011
VKTM started with pid=3, OS id=10331 at elevated priority
VKTM running at (20)ms precision
Fri Jun 19 17:05:37 2011
DIAG started with pid=4, OS id=10335
```

As shown in Example 13–1, query `V$DIAG_INFO` to locate the alert log.

Trace Files

A **trace file** is an administrative file that contain diagnostic data used to investigate problems. Also, trace files can provide guidance for tuning applications or an instance, as explained in "Performance Diagnostics and Tuning" on page 18-20.

Types of Trace Files

Each server and background process can periodically write to an associated trace file. The files information on the process environment, status, activities, and errors.

The SQL trace facility also creates trace files, which provide performance information on individual SQL statements. To enable tracing for a client identifier, service, module, action, session, instance, or database, you must execute the appropriate procedures in the `DBMS_MONITOR` package or use Oracle Enterprise Manager.

A **dump** is a special type of trace file. Whereas a trace tends to be continuous output of diagnostic data, a dump is typically a one-time output of diagnostic data in response to an event (such as an incident). When an incident occurs, the database writes one or more dumps to the incident directory created for the incident. Incident dumps also contain the incident number in the file name.

See Also:

- "Session Control Statements" on page 7-8
- *Oracle Database Administrator's Guide* to learn about trace files, dumps, and core files
- *Oracle Database Performance Tuning Guide* to learn about application tracing

Locations of Trace Files

ADR stores trace files in the `trace` subdirectory, as shown in Figure 13–8. Trace file names are platform-dependent and use the extension `.trc`.

Typically, database background process trace file names contain the Oracle SID, the background process name, and the operating system process number. An example of a trace file for the `RECO` process is `mytest_reco_10355.trc`.

Server process trace file names contain the Oracle SID, the string `ora`, and the operating system process number. An example of a server process trace file name is `mytest_ora_10304.trc`.

Sometimes trace files have corresponding trace map (`.trm`) files. These files contain structural information about trace files and are used for searching and navigation.

See Also: *Oracle Database Administrator's Guide* to learn how to find trace files

14

Memory Architecture

This chapter discusses the memory architecture of an Oracle Database **instance**.

This chapter contains the following sections:

- Introduction to Oracle Database Memory Structures
- Overview of the User Global Area
- Overview of the Program Global Area
- Overview of the System Global Area
- Overview of Software Code Areas

> **See Also:** *Oracle Database Administrator's Guide* for instructions for configuring and managing memory

Introduction to Oracle Database Memory Structures

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

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

> **See Also:** Chapter 15, "Process Architecture"

Basic Memory Structures

The basic memory structures associated with Oracle Database include:

- System global area (SGA)

  The SGA is a group of shared memory structures, known as **SGA components**, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

- Program global area (PGA)

A PGA is a nonshared memory region that contains data and control information exclusively for use by an Oracle process. The PGA is created by Oracle Database when an Oracle process is started.

One PGA exists for each **server process** and background process. The collection of individual PGAs is the **total instance PGA**, or **instance PGA**. Database initialization parameters set the size of the instance PGA, not individual PGAs.

- User Global Area (UGA)

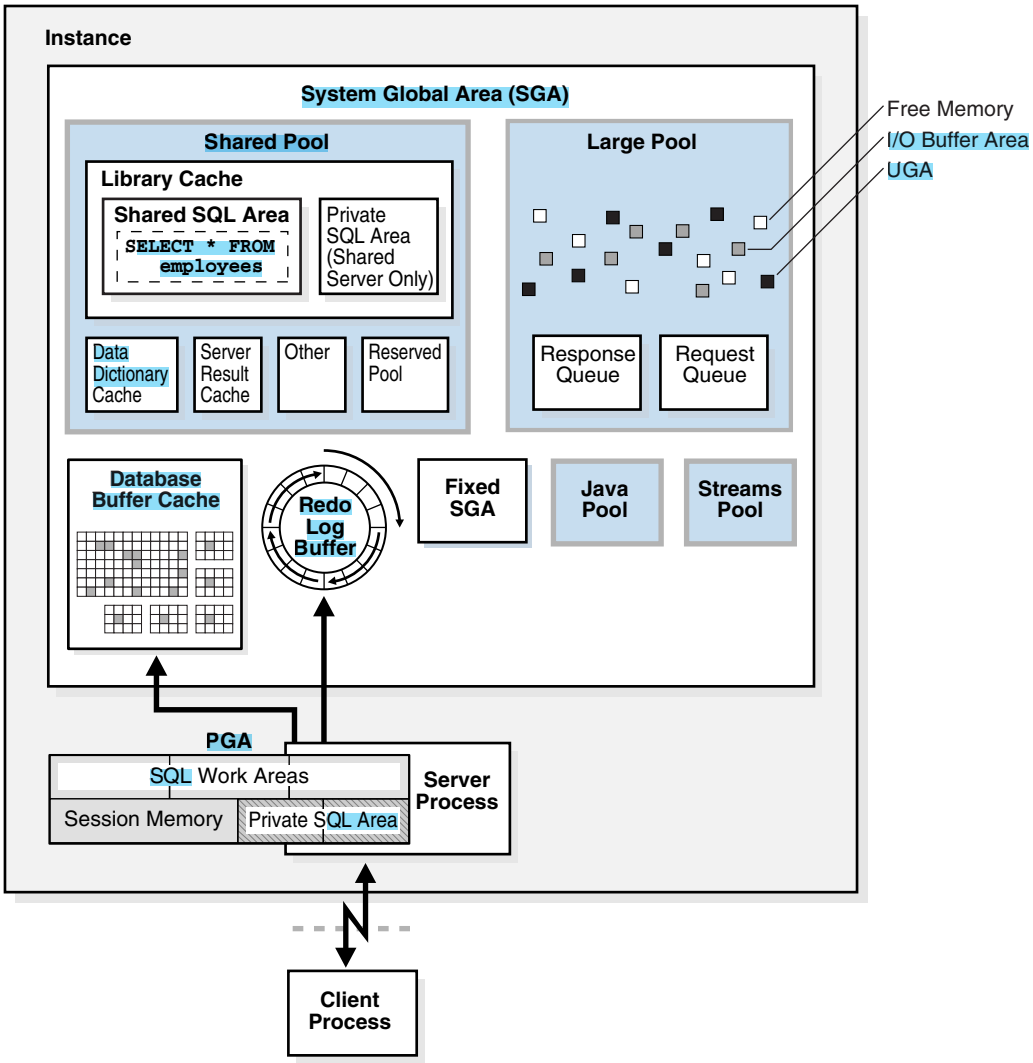
  The UGA is memory associated with a user session.

- Software code areas

  Software code areas are portions of memory used to store code that is being run or can be run. Oracle Database code is stored in a software area that is typically at a different location from user programs—a more exclusive or protected location.

Figure 14–1 illustrates the relationships among these memory structures.

Figure 14–1 Oracle Database Memory Structures



Oracle Database Memory Management

Memory management involves maintaining optimal sizes for the Oracle instance memory structures as demands on the database change. Oracle Database manages memory based on the settings of memory-related **initialization parameters**. The basic options for memory management are as follows:

- Automatic memory management

  You specify the target size for instance memory. The database instance automatically tunes to the target memory size, redistributing memory as needed between the SGA and the instance PGA.

- Automatic shared memory management

  This management mode is partially automated. You set a target size for the SGA and then have the option of setting an aggregate target size for the PGA or managing PGA work areas individually.

- Manual memory management

  Instead of setting the total memory size, you set many initialization parameters to manage components of the SGA and instance PGA individually.

If you create a database with Database Configuration Assistant (DBCA) and choose the basic installation option, then automatic memory management is the default.

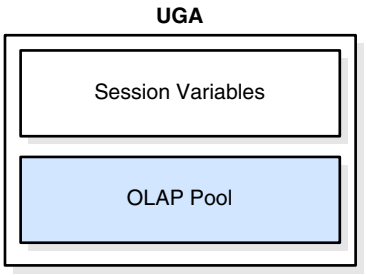
See Also:

- "Memory Management" on page 18-15 for more information about memory management options for DBAs
- "Tools for Database Installation and Configuration" to learn about DBCA
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn about memory management options

Overview of the User Global Area

The UGA is **session memory**, which is memory allocated for session variables, such as logon information, and other information required by a database session. Essentially, the UGA stores the session state. Figure 14–2 depicts the UGA.

Figure 14–2 User Global Area (UGA)



If a session loads a **PL/SQL package** into memory, then the UGA contains the **package state**, which is the set of values stored in all the package variables at a specific time (see "PL/SQL Packages" on page 8-6). The package state changes when a package subprogram changes the variables. By default, the package variables are unique to and persist for the life of the session.

The **OLAP page pool** is also stored in the UGA. This pool manages **OLAP** data pages, which are equivalent to data blocks. The page pool is allocated at the start of an OLAP session and released at the end of the session. An OLAP session opens automatically whenever a user queries a dimensional object such as a **cube**.

The UGA must be available to a database session for the life of the session. For this reason, the UGA cannot be stored in the PGA when using a **shared server** connection because the PGA is specific to a single process. Therefore, the UGA is stored in the SGA when using shared server connections, enabling any shared server process access to it. When using a **dedicated server** connection, the UGA is stored in the PGA.

See Also:

- "Connections and Sessions" on page 15-4
- *Oracle Database Net Services Administrator's Guide* to learn about shared server connections
- *Oracle OLAP User's Guide* for an overview of Oracle OLAP

Overview of the Program Global Area

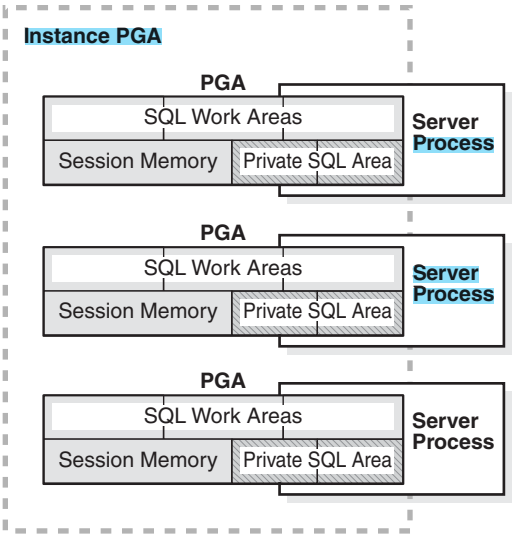
The PGA is memory specific to an operating process or thread that is not shared by other processes or threads on the system. Because the PGA is process-specific, it is never allocated in the SGA.

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 14–3 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 (see "Summary of Memory Management Methods" on page 18-17). Individual PGAs can grow as needed up to this target size.

Figure 14–3 Instance PGA

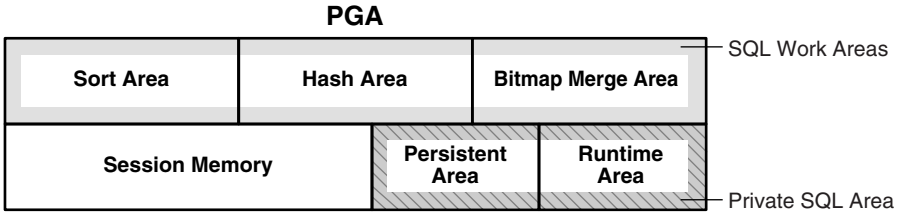


Note: Background processes also allocate their own PGAs. This discussion focuses on server process PGAs only.

Contents of the PGA

The PGA is subdivided into different areas, each with a different purpose. Figure 14–4 shows the possible contents of the PGA for a dedicated server session. Not all of the PGA areas will exist in every case.

Figure 14–4 PGA Contents



Private SQL Area

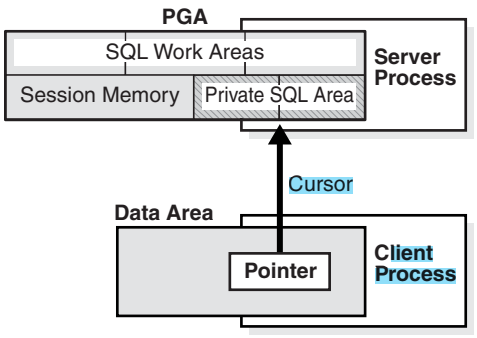
A **private SQL area** holds information about a parsed SQL statement and other session-specific information for processing. When a server process executes SQL or PL/SQL code, the process uses the private SQL area to store **bind variable** values, query execution state information, and query execution work areas.

Do not confuse a *private* SQL area, which is in the UGA, with the *shared* SQL area, which stores execution plans in the SGA. Multiple private SQL areas in the same or different sessions can point to a single execution plan in the SGA. For example, 20 executions of `SELECT * FROM employees` in one session and 10 executions of the same query in a different session can share the same plan. The private SQL areas for each execution are not shared and may contain different values and data.

A **cursor** is a name or handle to a specific private SQL area. As shown in Figure 14–5, you can think of a cursor as a pointer on the client side and as a state on the server

side. Because cursors are closely associated with private SQL areas, the terms are sometimes used interchangeably.

Figure 14–5 Cursor



A private SQL area is divided into the following areas:

- The run-time area

  This area contains query execution state information. For example, the run-time area tracks the number of rows retrieved so far in a **full table scan**.

  Oracle Database creates the run-time area as the first step of an execute request. For **DML** statements, the run-time area is freed when the SQL statement is closed.

- The persistent area

  This area contains **bind variable** values. A bind variable value is supplied to a SQL statement at run time when the statement is executed. The persistent area is freed only when the cursor is closed.

The client process is responsible for managing private SQL areas. The allocation and deallocation of private SQL areas depends largely on the application, although the number of private SQL areas that a client process can allocate is limited by the initialization parameter `OPEN_CURSORS`.

Although most users rely on the automatic cursor handling of database utilities, the Oracle Database programmatic interfaces offer developers more control over cursors. In general, applications should close all open cursors that will not be used again to free the persistent area and to minimize the memory required for application users.

See Also:

- "Shared SQL Areas" on page 14-16
- *Oracle Database Advanced Application Developer's Guide* and *Oracle Database PL/SQL Language Reference* to learn how to use cursors

SQL Work Areas

A **work area** is a private allocation of PGA memory used for memory-intensive operations. For example, a sort operator uses the **sort area** to sort a set of rows. Similarly, a **hash join** operator uses a **hash area** to build a **hash table** from its left input, whereas a **bitmap merge** uses the **bitmap merge area** to merge data retrieved from scans of multiple bitmap indexes.

Example 14–1 shows a **join** of `employees` and `departments` with its **query plan**.

Example 14–1 Query Plan for Table Join

```
SQL> SELECT *
  2  FROM   employees e JOIN departments d
  3  ON     e.department_id=d.department_id
  4  ORDER BY last_name;
.
.
.
------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |   106 |  9328 |     7  (29)| 00:00:01 |
|   1 |  SORT ORDER BY      |             |   106 |  9328 |     7  (29)| 00:00:01 |
|*  2 |   HASH JOIN         |             |   106 |  9328 |     6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| DEPARTMENTS |    27 |   540 |     2   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| EMPLOYEES   |   107 |  7276 |     3   (0)| 00:00:01 |
------------------------------------------------------------------------------
```

In Example 14–1, the run-time area tracks the progress of the full table scans. The session performs a hash join in the hash area to match rows from the two tables. The `ORDER BY` sort occurs in the sort area.

If the amount of data to be processed by the operators does not fit into a work area, then Oracle Database divides the input data into smaller pieces. In this way, the database processes some data pieces in memory while writing the rest to temporary disk storage for processing later.

The database automatically tunes work area sizes when automatic PGA memory management is enabled. You can also manually control and tune the size of a work area. See "Memory Management" on page 18-15 for more information.

Generally, larger work areas can significantly improve performance of an operator at the cost of higher memory consumption. Optimally, the size of a work area is sufficient to accommodate the input data and auxiliary memory structures allocated by its associated SQL operator. If not, response time increases because part of the input data must be cached on disk. In the extreme case, if the size of a work area is too small compared to input data size, then the database must perform multiple passes over the data pieces, dramatically increasing response time.

See Also:

- *Oracle Database Administrator's Guide* to learn how to use automatic PGA management
- *Oracle Database Performance Tuning Guide* to learn how to tune PGA memory

PGA Usage in Dedicated and Shared Server Modes

PGA memory allocation depends on whether the database uses dedicated or shared server connections. Table 14–1 shows the differences.

Table 14–1 Differences in Memory Allocation Between Dedicated and Shared Servers

Memory Area	Dedicated Server	Shared Server
Nature of session memory	Private	Shared
Location of the persistent area	PGA	SGA
Location of the run-time area for DML/DDL statements	PGA	PGA

See Also: *Oracle Database Net Services Administrator's Guide* to learn how to configure a database for shared server

Overview of the System Global Area

The **SGA** is a read/write memory area that, along with the Oracle background processes, make up a database instance. All server processes that execute on behalf of users can read information in the instance SGA. Several processes write to the SGA during database operation.

Note: The server and background processes do not reside *within* the SGA, but exist in a separate memory space.

Each database instance has its own SGA. Oracle Database automatically allocates memory for an SGA at instance startup and reclaims the memory at instance shutdown. When you start an instance with SQL*Plus or Oracle Enterprise Manager, the size of the SGA is shown as in the following example:

```
SQL> STARTUP
ORACLE instance started.

Total System Global Area  368283648 bytes
Fixed Size                  1300440 bytes
Variable Size             343935016 bytes
Database Buffers           16777216 bytes
Redo Buffers                6270976 bytes
Database mounted.
Database opened.
```

As shown in Figure 14–1, the SGA consists of several **memory components**, which are pools of memory used to satisfy a particular class of memory allocation requests. All SGA components except the redo log buffer allocate and deallocate space in units of contiguous memory called **granules**. Granule size is platform-specific and is determined by total SGA size.

You can query the `V$SGASTAT` view for information about SGA components.

The most important SGA components are the following:

- Database Buffer Cache
- Redo Log Buffer
- Shared Pool
- Large Pool
- Java Pool
- Streams Pool
- Fixed SGA

See Also:

- "Introduction to the Oracle Database Instance" on page 13-1
- *Oracle Database Performance Tuning Guide* to learn more about granule sizing

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.

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

- 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 disk but does not immediately write data blocks to disk. Instead, **database writer (DBW)** performs **lazy writes** in the background.

- 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 a **flash disk device**, which is a solid state storage device that uses flash memory. The database can improve performance by caching buffers in flash memory instead of reading from magnetic disk.

> **Note:** Database Smart Flash Cache is available only in Solaris and Oracle Enterprise Linux.

Buffer States

The database uses internal algorithms to manage buffers in the cache. A buffer can be in any of the following mutually exclusive states:

- Unused

  The buffer is available for use because it has never been used or is currently unused. This type of buffer is the easiest for the database to use.

- Clean

  This buffer was used earlier and now contains a read-consistent version of a block as of a point in time. The block contains data but is "clean" so it does not need to be checkpointed. The database can pin the block and reuse it.

- Dirty

  The buffer contain modified data that has not yet been written to disk. The database must checkpoint the block before reusing it.

Every buffer has an access mode: **pinned** or **free** (unpinned). A buffer is "pinned" in the cache so that it does not age out of memory while a user session accesses it. Multiple sessions cannot modify a pinned buffer at the same time.

The database uses a sophisticated algorithm to make buffer access efficient. Pointers to dirty and nondirty buffers exist on the same **least recently used (LRU) list**, which has a hot end and cold end. A **cold buffer** is one that has not been recently used. A **hot buffer** is frequently accessed and has been recently used.

> **Note:** Conceptually, there is only one LRU, but for **concurrency** the database actually uses several LRUs.

Buffer Modes

When a client requests data, Oracle Database retrieves buffers from the database buffer cache in either of the following modes:

- Current mode

  A **current mode get**, also called a **db block get**, is a retrieval of a block as it currently appears in the buffer cache. For example, if an uncommitted transaction has updated two rows in a block, then a current mode get retrieves the block with these uncommitted rows. The database uses db block gets most frequently during modification statements, which must update only the current version of the block.

- Consistent mode

  A **consistent read get** is a retrieval of a read-consistent version of a block. This retrieval may use **undo data**. For example, if an uncommitted transaction has updated two rows in a block, and if a query in a separate session requests the block, then the database uses undo data to create a read-consistent version of this block (called a **consistent read clone**) that does not include the uncommitted updates. Typically, a query retrieves blocks in consistent mode.

  See Also:

  - "Read Consistency and Undo Segments" on page 9-3
  - *Oracle Database Reference* for descriptions of database statistics such as db block get and consistent read get

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. When a requested buffer is not found in memory, the database performs a **physical I/O** to copy the buffer from either the flash cache or disk into memory, and then a logical I/O to read the cached buffer.

Buffer Writes The **database writer (DBW)** process periodically writes cold, dirty buffers to disk. DBW*n* writes buffers in the following circumstances:

- A server process cannot find clean buffers for reading new blocks into the database buffer cache.

  As buffers are dirtied, the number of free buffers decreases. If the number drops below an internal threshold, and if clean buffers are required, then server processes signal DBW*n* to write.

  The database uses the LRU to determine which dirty buffers to write. When dirty buffers reach the cold end of the LRU, the database moves them off the LRU to a **write queue**. DBW*n* writes buffers in the queue to disk, using multiblock writes if possible. This mechanism prevents the end of the LRU from becoming clogged with dirty buffers and allows clean buffers to be found for reuse.

- The database must advance the **checkpoint**, which is the position in the redo thread from which **instance recovery** must begin.

- Tablespaces are changed to read-only status or taken offline.

  See Also:

  - "Database Writer Process (DBWn)" on page 15-8
  - *Oracle Database Performance Tuning Guide* to learn how to diagnose and tune buffer write issues

Buffer Reads When the number of clean or unused buffers is low, the database must remove buffers from the buffer cache. The algorithm depends on whether the flash cache is enabled:

- Flash cache disabled

  The database re-uses each clean buffer as needed, overwriting it. If the overwritten buffer is needed later, then the database must read it from magnetic disk.

- Flash cache enabled

  DBW*n* can write the body of a clean buffer to the flash cache, enabling reuse of its in-memory buffer. The database keeps the buffer header in an LRU list in main memory to track the state and location of the buffer body in the flash cache. If this buffer is needed later, then the database can read it from the flash cache instead of from magnetic disk.

When a client process requests a buffer, the server process searches the buffer cache for the buffer. A **cache hit** occurs if the database finds the buffer in memory. The search order is as follows:

1. The server process searches for the whole buffer in the buffer cache.

   If the process finds the whole buffer, then the database performs a **logical read** of this buffer.

2. The server process searches for the buffer header in the flash cache LRU list.

   If the process finds the buffer header, then the database performs an **optimized physical read** of the buffer body from the flash cache into the in-memory cache.

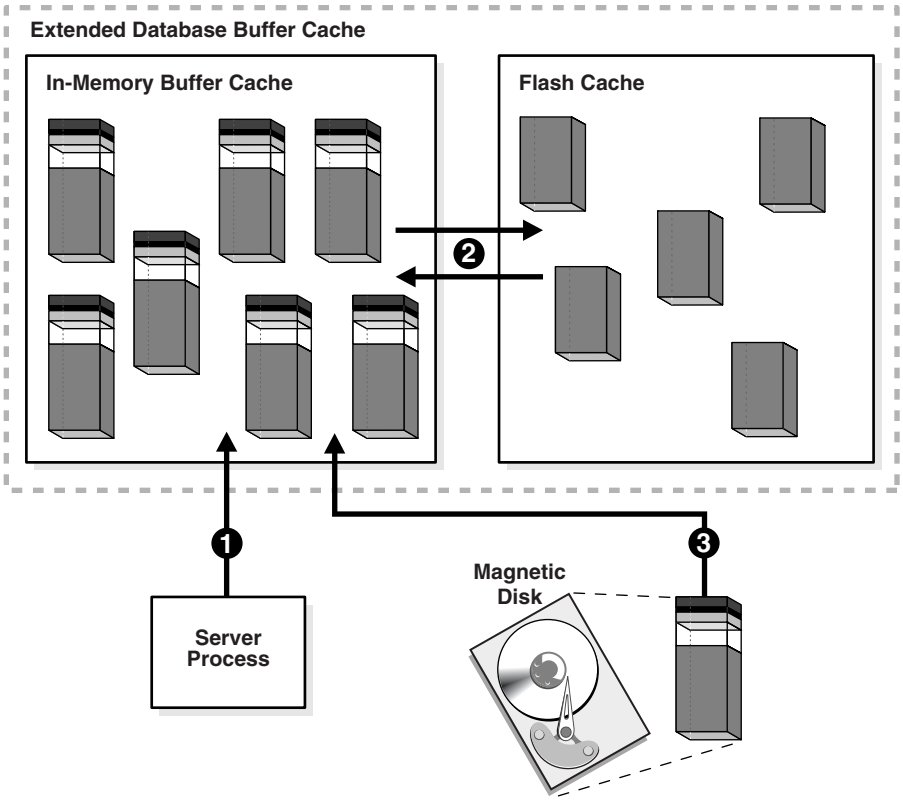
3. If the process does *not* find the buffer in memory (a **cache miss**), then the server process performs the following steps:

   a. Copies the block from a data file into memory (a **physical read**)

   b. Performs a logical read of the buffer that was read into memory

Figure 14–6 illustrates the buffer search order. The extended buffer cache includes both the in-memory buffer cache, which contains whole buffers, and the flash cache, which contains buffer bodies. In the figure, the database searches for a buffer in the buffer cache and, not finding the buffer, reads it into memory from magnetic disk.

Figure 14–6 Buffer Search



In general, accessing data through a cache hit is faster than through a cache miss. The **buffer cache hit ratio** measures how often the database found a requested block in the buffer cache without needing to read it from disk.

The database can perform physical reads from either a data file or a **temp file**. Reads from a data file are followed by logical I/Os. Reads from a temp file occur when insufficient memory forces the database write data to a **temporary table** and read it back later. These physical reads bypass the buffer cache and do not incur a logical I/O.

> **See Also:** *Oracle Database Performance Tuning Guide* to learn how to calculate the buffer cache hit ratio

Buffer Touch Counts The database measures the frequency of access of buffers on the LRU list using a **touch count**. This mechanism enables the database to increment a counter when a buffer is pinned instead of constantly shuffling buffers on the LRU list.

> **Note:** The database does not physically move blocks in memory. The movement is the change in location of a pointer on a list.

When a buffer is pinned, the database determines when its touch count was last incremented. If the count was incremented over three seconds ago, then the count is incremented; otherwise, the count stays the same. The three-second rule prevents a burst of pins on a buffer counting as many touches. For example, a session may insert several rows in a data block, but the database considers these inserts as one touch.

If a buffer is on the cold end of the LRU, but its touch count is high, then the buffer moves to the hot end. If the touch count is low, then the buffer ages out of the cache.

Buffers and Full Table Scans When buffers must be read from disk, the database inserts the buffers into the middle of the LRU list. In this way, hot blocks can remain in the cache so that they do not need to be read from disk again.

A problem is posed by a **full table scan**, which sequentially reads all rows under the table **high water mark** (see "Segment Space and the High Water Mark" on page 12-27). Suppose that the total size of the blocks in a table segment is greater than the size of the buffer cache. A full scan of this table could clean out the buffer cache, preventing the database from maintaining a cache of frequently accessed blocks.

Blocks read into the database cache as the result of a full scan of a large table are treated differently from other types of reads. The blocks are immediately available for reuse to prevent the scan from effectively cleaning out the buffer cache.

In the rare case where the default behavior is not desired, you can change the `CACHE` attribute of the table. In this case, the database does not force or pin the blocks in the buffer cache, but ages them out of the cache in the same way as any other block. Use care when exercising this option because a full scan of a large table may clean most of the other blocks out of the cache.

See Also:

- *Oracle Database SQL Language Reference* for information about the `CACHE` clause
- *Oracle Database Performance Tuning Guide* to learn how to interpret buffer cache advisory statistics

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:

- 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.

- 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.

- Recycle pool

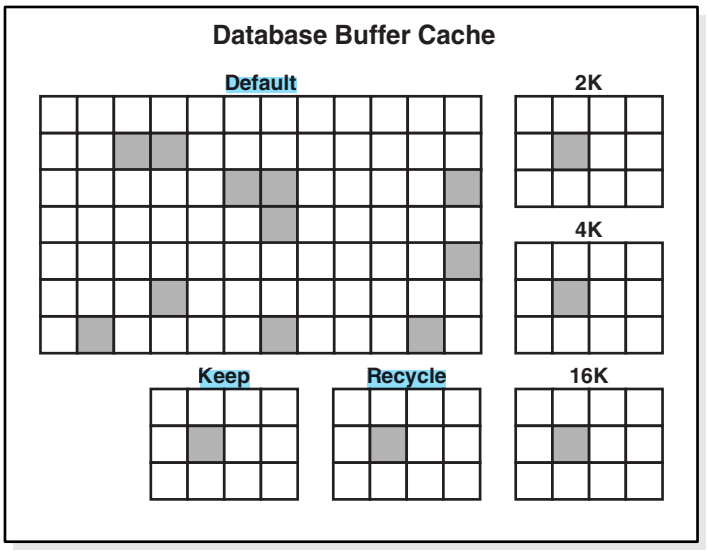
  This pool is intended for blocks that are used infrequently. A recycle pool prevent objects from consuming unnecessary space in the cache.

A database has a standard block size (see "Database Block Size" on page 12-7). You can create a tablespace with a block size that differs from the standard size. Each nondefault block size has its own pool. Oracle Database manages the blocks in these pools in the same way as in the default pool.

Figure 14–7 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 14–7 Database Buffer Cache



See Also:

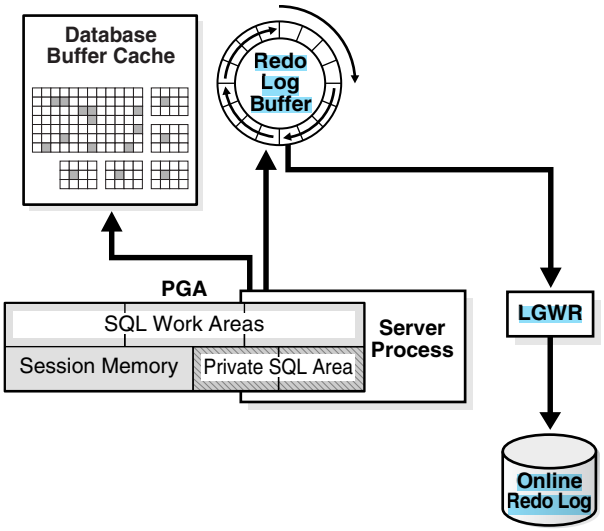
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn more about buffer pools
- *Oracle Database Performance Tuning Guide* to learn how to use multiple buffer pools

Redo Log Buffer

The **redo log buffer** is a circular buffer in the SGA that stores redo entries describing changes made to the database. **Redo entries** contain the information necessary to reconstruct, or redo, changes made to the database by DML or DDL operations. Database recovery applies redo entries to data files to reconstruct lost changes.

Oracle Database processes copy redo entries from the user memory space to the redo log buffer in the SGA. The redo entries take up continuous, sequential space in the buffer. The background process **log writer (LGWR)** writes the redo log buffer to the active online redo log group on disk. Figure 14–8 shows this redo buffer activity.

Figure 14–8 Redo Log Buffer



LGWR writes redo sequentially to disk while DBW*n* performs scattered writes of data blocks to disk. Scattered writes tend to be much slower than sequential writes. Because LGWR enable users to avoid waiting for DBW*n* to complete its slow writes, the database delivers better performance.

The `LOG_BUFFER` initialization parameter specifies the amount of memory that Oracle Database uses when buffering redo entries. Unlike other SGA components, the redo log buffer and fixed SGA buffer do not divide memory into granules.

See Also:

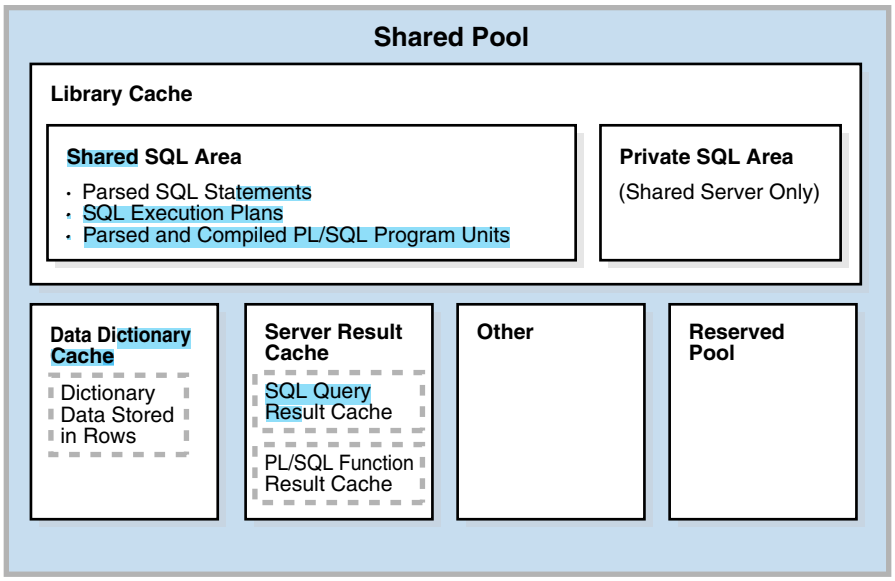
- "Log Writer Process (LGWR)" on page 15-9 and "Importance of Checkpoints for Instance Recovery" on page 13-13
- *Oracle Database Administrator's Guide* for information about the online redo log

Shared Pool

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, the most important of which are shown in Figure 14–9.

Figure 14–9 Shared Pool



This section includes the following topics:

- Library Cache
- Data Dictionary Cache
- Server Result Cache
- Reserved Pool

Library Cache

The **library cache** is a shared pool memory structure that stores executable SQL and PL/SQL code. This cache contains the shared SQL and PL/SQL areas and control structures such as locks and library cache handles. In a shared server architecture, the library cache also contains private SQL areas.

When a SQL statement is executed, the database attempts to reuse previously executed code. If a parsed representation of a SQL statement exists in the library cache and can be shared, then the database reuses the code, known as a **soft parse** or a **library cache hit**. Otherwise, the database must build a new executable version of the application code, known as a **hard parse** or a **library cache miss**.

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

- 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.

- 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:

   - 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.
   - 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**.

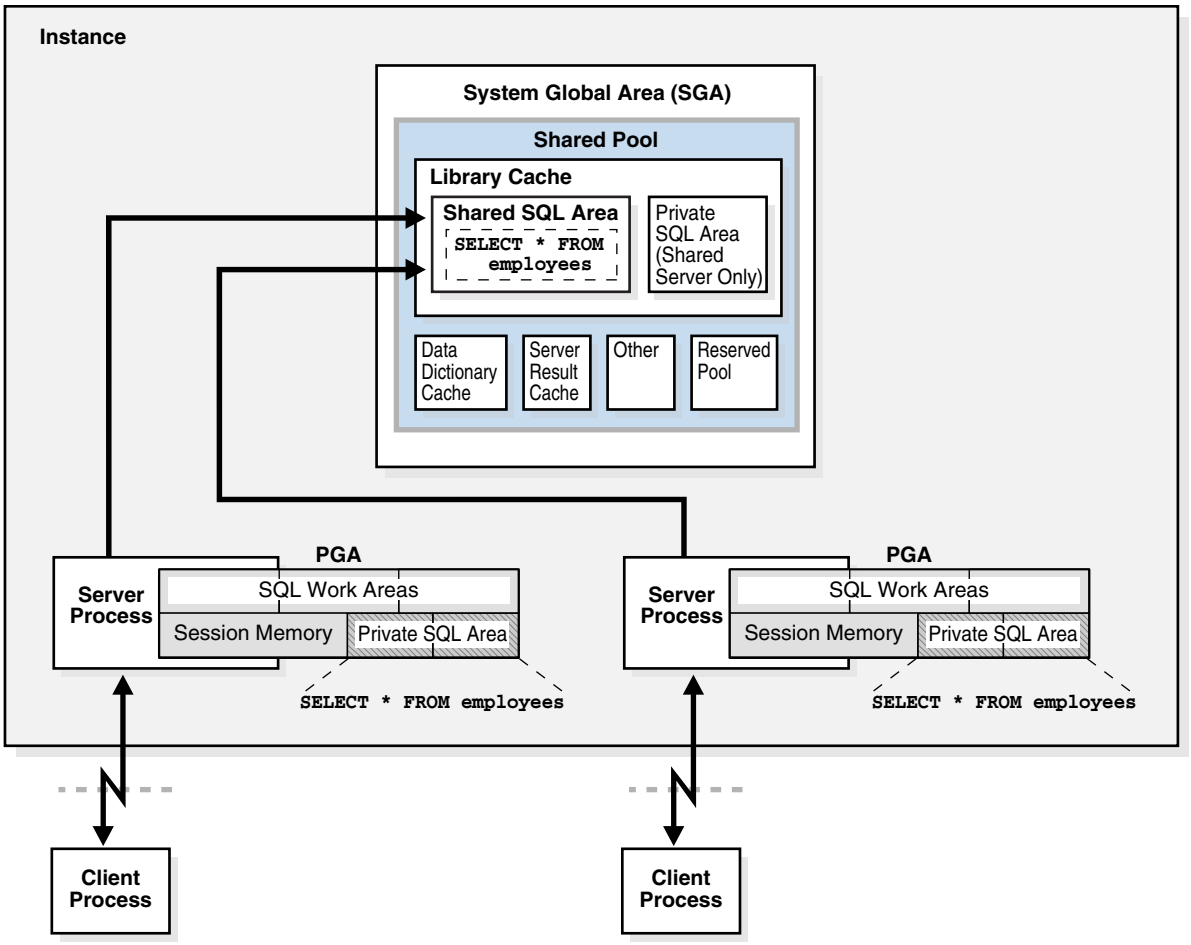
   In either case, the private SQL area for the user points to the shared SQL area that contains the statement and execution plan.

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 14–10 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 14–10 Private SQL Areas and Shared SQL Area



See Also:

- *Oracle Database Performance Tuning Guide* to learn more about managing the library cache
- *Oracle Database Advanced Application Developer's Guide* for more information about shared SQL

Program Units and the Library Cache The library cache holds executable forms of PL/SQL programs and Java classes. These items are collectively referred to as **program units**.

The database processes program units similarly to SQL statements. For example, the database allocates a shared area to hold the parsed, compiled form of a PL/SQL program. The database allocates a private area to hold values specific to the session that runs the program, including local, global, and package variables, and buffers for executing SQL. If multiple users run the same program, then each user maintains a separate copy of his or her private SQL area, which holds session-specific values, and accesses a single shared SQL area.

The database processes individual SQL statements within a PL/SQL program unit as previously described. Despite their origins within a PL/SQL program unit, these SQL statements use a shared area to hold their parsed representations and a private area for each session that runs the statement.

Allocation and Reuse of Memory in the Shared Pool The database allocates shared pool memory when a new SQL statement is parsed. The memory size depends on the complexity of the statement.

In general, an item in the shared pool stays until it is removed according to an LRU algorithm. The database allows shared pool items used by many sessions to remain in memory as long as they are useful, even if the process that created the item terminates. This mechanism minimizes the overhead and processing of SQL statements.

If space is needed for new items, then the database frees memory for infrequently used items. A shared SQL area can be removed from the shared pool even if the shared SQL area corresponds to an open cursor that has not been used for some time. If the open cursor is subsequently used to run its statement, then Oracle Database reparses the statement and allocates a new shared SQL area.

The database also removes a shared SQL area from the shared pool in the following circumstances:

- If statistics are gathered for a table, **table cluster**, or index, then by default the database gradually removes all shared SQL areas that contain statements referencing the analyzed object after a period of time. The next time a removed statement is run, the database parses it in a new shared SQL area to reflect the new statistics for the schema object.
- If a schema object is referenced in a SQL statement, and if this object is later modified by a DDL statement, then the database invalidates the shared SQL area. The optimizer must reparse the statement the next time it is run.
- If you change the global database name, then the database removes all information from the shared pool.

You can use the `ALTER SYSTEM FLUSH SHARED_POOL` statement to manually remove all information in the shared pool to assess the performance that can be expected after an instance restart.

See Also:

- *Oracle Database SQL Language Reference* for information about using `ALTER SYSTEM FLUSH SHARED_POOL`
- *Oracle Database Reference* for information about `V$SQL` and `V$SQLAREA` dynamic views

Data Dictionary Cache

The **data dictionary** is a collection of database tables and views containing reference information about the database, its structures, and its users. Oracle Database accesses the data dictionary frequently during SQL statement parsing.

The data dictionary is accessed so often by Oracle Database that the following special memory locations are designated to hold dictionary data:

- Data dictionary cache

  This cache holds information about database objects. The cache is also known as the **row cache** because it holds data as rows instead of buffers.

- Library cache

All server processes share these caches for access to data dictionary information.

See Also:

- Chapter 6, "Data Dictionary and Dynamic Performance Views"
- *Oracle Database Performance Tuning Guide* to learn how to allocate additional memory to the data dictionary cache

Server Result Cache

Unlike the buffer pools, the **server result cache** holds result sets and not data blocks. The server result cache contains the **SQL query result cache** and **PL/SQL function result cache**, which share the same infrastructure.

A **client result cache** differs from the server result cache. A client cache is configured at the application level and is located in client memory, not in database memory.

See Also:

- *Oracle Database Administrator's Guide* for information about sizing the result cache
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_RESULT_CACHE` package
- *Oracle Database Performance Tuning Guide* for more information about the client result cache

SQL Query Result Cache The database can store the results of queries and query fragments in the **SQL query result cache**, using the cached results for future queries and query fragments. Most applications benefit from this performance improvement.

For example, suppose an application runs the same `SELECT` statement repeatedly. If the results are cached, then the database returns them immediately. In this way, the database avoids the expensive operation of rereading blocks and recomputing results. The database automatically invalidates a cached result whenever a transaction modifies the data or metadata of database objects used to construct that cached result.

Users can annotate a query or query fragment with a `RESULT_CACHE` **hint** to indicate that the database should store results in the SQL query result cache. The `RESULT_CACHE_MODE` initialization parameter determines whether the SQL query result cache is used for all queries (when possible) or only for annotated queries.

See Also:

- *Oracle Database Reference* to learn more about the `RESULT_CACHE_MODE` initialization parameter
- *Oracle Database SQL Language Reference* to learn about the `RESULT_CACHE` hint

PL/SQL Function Result Cache The **PL/SQL function result cache** stores function result sets. Without caching, 1000 calls of a function at 1 second per call would take 1000 seconds. With caching, 1000 function calls with the same inputs could take 1 second *total*. Good candidates for result caching are frequently invoked functions that depend on relatively static data.

PL/SQL function code can include a request to cache its results. Upon invocation of this function, the system checks the cache. If the cache contains the result from a previous function call with the same parameter values, then the system returns the cached result to the invoker and does not reexecute the function body. If the cache

does not contain the result, then the system executes the function body and adds the result (for these parameter values) to the cache before returning control to the invoker.

> **Note:** You can specify the database objects that are used to compute a cached result so that if any of them are updated, the cached result becomes invalid and must be recomputed.

The cache can accumulate many results—one result for every unique combination of parameter values with which each result-cached function was invoked. If the database needs more memory, then it ages out one or more cached results.

See Also:

- *Oracle Database Advanced Application Developer's Guide* to learn more about the PL/SQL function result cache
- *Oracle Database PL/SQL Language Reference* to learn more about the PL/SQL function result cache

Reserved Pool

The **reserved pool** is a memory area in the shared pool that Oracle Database can use to allocate large contiguous chunks of memory.

Allocation of memory from the shared pool is performed in chunks. Chunking allows large objects (over 5 KB) to be loaded into the cache without requiring a single contiguous area. In this way, the database reduces the possibility of running out of contiguous memory because of fragmentation.

Infrequently, Java, PL/SQL, or SQL cursors may make allocations out of the shared pool that are larger than 5 KB. To allow these allocations to occur most efficiently, the database segregates a small amount of the shared pool for the reserved pool.

See Also: *Oracle Database Performance Tuning Guide* to learn how to configure the reserved pool

Large Pool

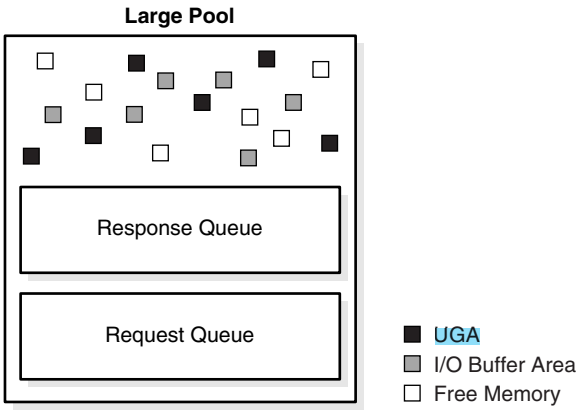
The **large pool** is an optional memory area intended for memory allocations that are larger than is appropriate for the shared pool. The large pool can provide large memory allocations for the following:

- UGA for the shared server and the **Oracle XA** interface (used where transactions interact with multiple databases)
- Message buffers used in the parallel execution of statements
- 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.

Figure 14–11 is a graphical depiction of the large pool.

Figure 14–11 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.

See Also:

- "Dispatcher Request and Response Queues" on page 16-12 to learn about allocating session memory for shared server
- *Oracle Database Advanced Application Developer's Guide* to learn about Oracle XA
- *Oracle Database Performance Tuning Guide* for more information about the large pool
- "Parallel Execution" on page 15-15 for information about allocating memory for parallel execution

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.

See Also:

- *Oracle Database Java Developer's Guide*
- *Oracle Database Performance Tuning Guide* to learn about views containing Java pool advisory statistics

Streams Pool

The **Streams pool** stores buffered queue messages and provides memory for Oracle Streams capture processes and apply processes. The Streams pool is used exclusively by Oracle Streams.

Unless you specifically configure it, the size of the Streams pool starts at zero. The pool size grows dynamically as required by Oracle Streams.

> **See Also:** *Oracle Database 2 Day + Data Replication and Integration Guide* and *Oracle Streams Replication Administrator's Guide*

Fixed SGA

The **fixed SGA** is an internal housekeeping area. For example, the fixed SGA contains:

- General information about the state of the database and the instance, which the background processes need to access
- Information communicated between processes, such as information about **locks** (see "Overview of Automatic Locks" on page 9-17)

The size of the fixed SGA is set by Oracle Database and cannot be altered manually. The fixed SGA size can change from release to release.

Overview of Software Code Areas

Software code areas are portions of memory that store code that is being run or can be run. Oracle Database code is stored in a software area that is typically more exclusive and protected than the location of user programs.

Software areas are usually static in size, changing only when software is updated or reinstalled. The required size of these areas varies by operating system.

Software areas are read-only and can be installed shared or nonshared. Some database tools and utilities, such as Oracle Forms and SQL*Plus, can be installed shared, but some cannot. When possible, database code is shared so that all users can access it without having multiple copies in memory, resulting in reduced main memory and overall improvement in performance. Multiple instances of a database can use the same database code area with different databases if running on the same computer.

> **Note:** The option of installing software shared is not available for all operating systems, for example, on PCs operating Windows. See your operating system-specific documentation for more information.