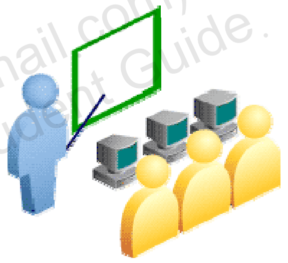
2

Exploring Oracle Database Architecture

Objectives

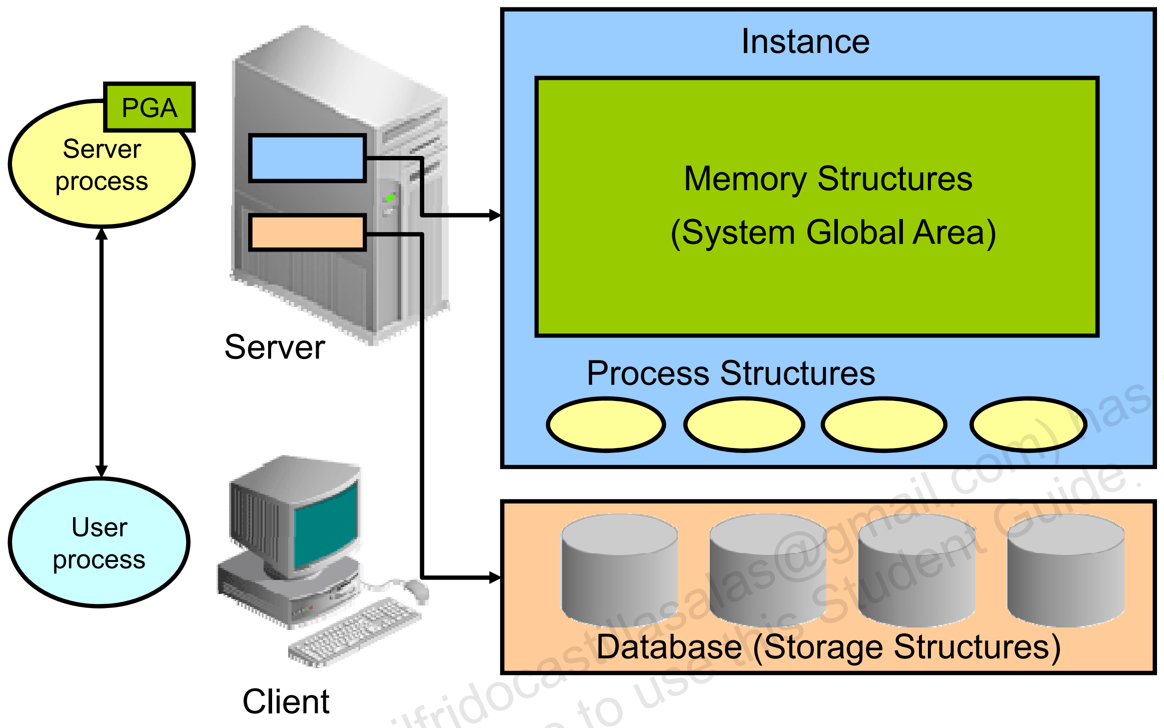
After completing this lesson, you should be able to:

- List the major architectural components of Oracle Database
- Explain memory structures
- Describe background processes
- Correlate logical and physical storage structures
- Describe pluggable databases
- Describe ASM storage components



This lesson provides a detailed overview of Oracle Database architecture. You learn about physical and logical structures and about various components.

Oracle Database Server Architecture: Overview



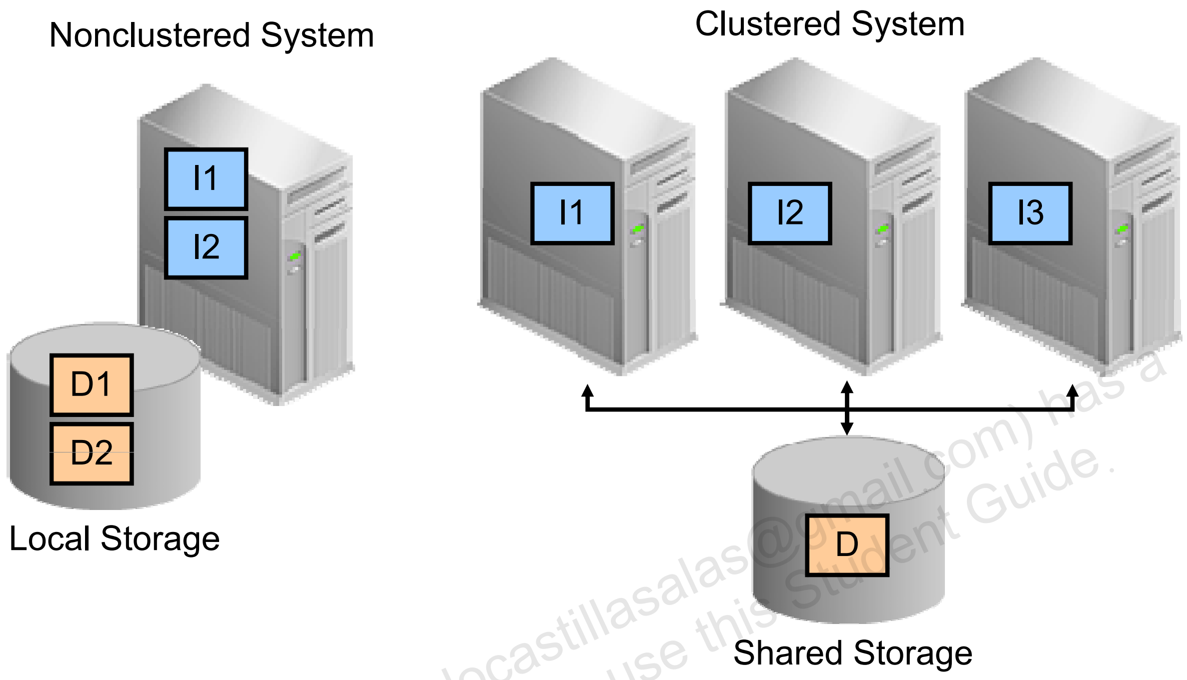
There are three major structures in Oracle Database server architecture: memory structures, process structures, and storage structures. A basic Oracle database system consists of an Oracle database and a database instance.

The database consists of both physical structures and logical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures.

The instance consists of memory structures and background processes associated with that instance. Every time an instance is started, a shared memory area called the System Global Area (SGA) is allocated and the background processes are started. Processes are jobs that work in the memory of computers. A process is defined as a “thread of control” or a mechanism in an operating system that can run a series of steps. After starting a database instance, the Oracle software associates the instance with a specific database. This is called *mounting the database*. The database is then ready to be opened, which makes it accessible to authorized users.

Note: Oracle Automatic Storage Management (ASM) uses the concept of an instance for the memory and process components, but is not associated with a specific database.

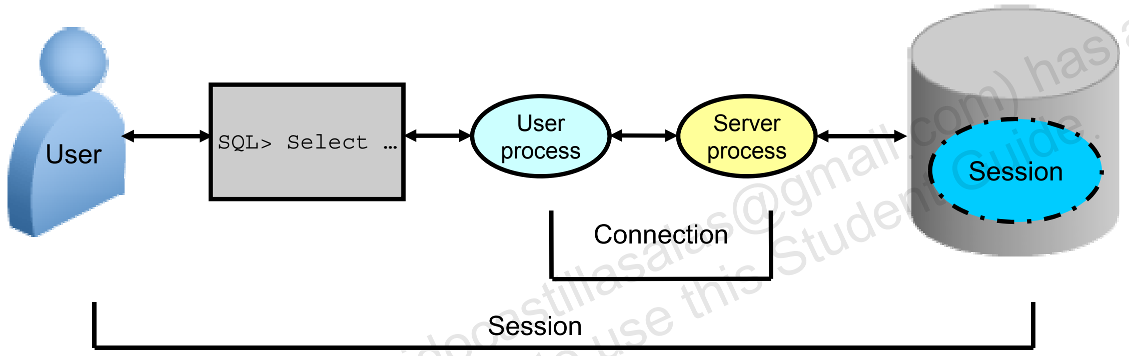
Oracle Database Instance Configurations



Each database instance is associated with one and only one database. If there are multiple databases on the same server, then there is a separate and distinct database instance for each database. A database instance cannot be shared. A Real Applications Cluster (RAC) database usually has multiple instances on separate servers for the same shared database. In this model, the same database is associated with each RAC instance, which meets the requirement that, at most, only one database is associated with an instance.

Connecting to the Database Instance

- Connection: Communication between a user process and an instance
- Session: Specific connection of a user to an instance through a user process



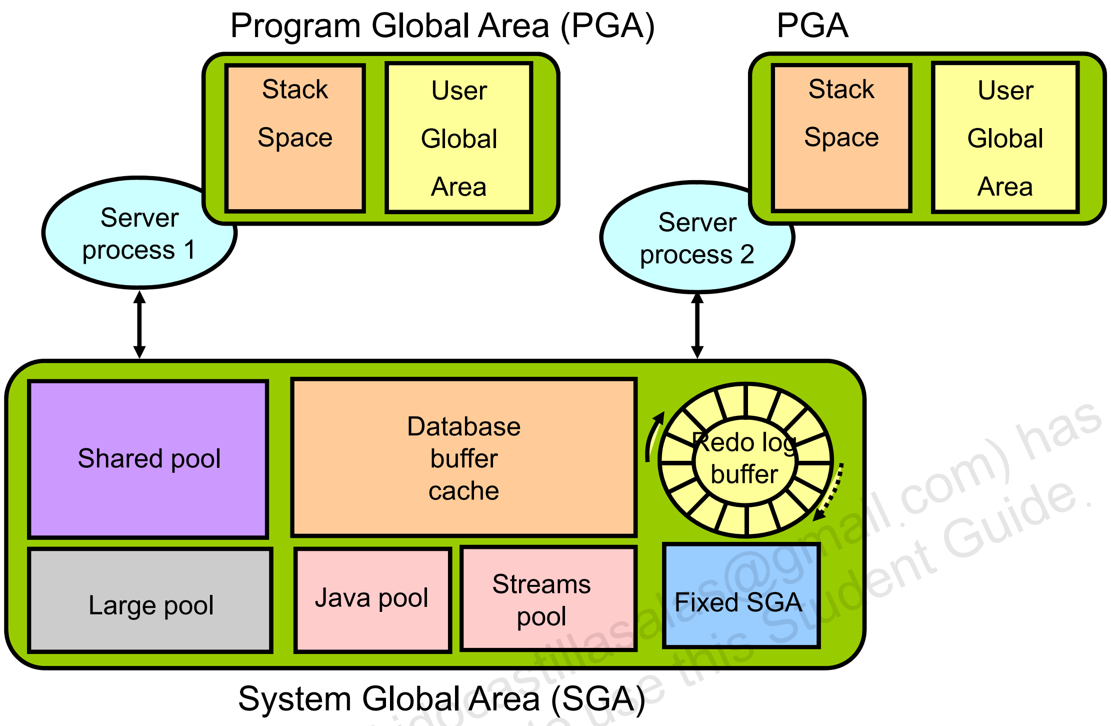
Connections and sessions are closely related to user processes but are very different in meaning.

A *connection* is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established using available interprocess communication mechanisms (on a computer that runs both the user process and Oracle Database) or network software (when different computers run the database application and Oracle Database, and communicate through a network).

A *session* represents the state of a current user login to the database instance. For example, when a user starts SQL*Plus, the user must provide a valid username and password, and then a session is established for that user. A session lasts from the time a user connects until the user disconnects or exits the database application.

Multiple sessions can be created and exist concurrently for a single Oracle database user using the same username. For example, a user with the username/password of `HR`/`HR` can connect to the same Oracle Database instance several times.

Oracle Database Memory Structures



Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data that is shared among users, and private data areas for each connected user.

Two basic memory structures are associated with an instance:

- **System Global Area (SGA):** Group of shared memory structures, known as SGA components, which 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 Areas (PGA):** Memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

The SGA is the memory area that contains data and control information for the instance. The SGA includes the following data structures:

- **Shared pool:** Caches various constructs that can be shared among users
- **Database buffer cache:** Caches blocks of data retrieved from the database
- **Redo log buffer:** Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk
- **Large pool:** Optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and I/O server processes
- **Java pool:** Used for all session-specific Java code and data in the Java Virtual Machine (JVM)
- **Streams pool:** Used by Oracle Streams to store information required by capture and apply
- **Fixed SGA:** An internal housekeeping area containing general information about the state of the database and the instance, and information communicated between processes

When you start the instance, the amount of memory allocated for the SGA is displayed.

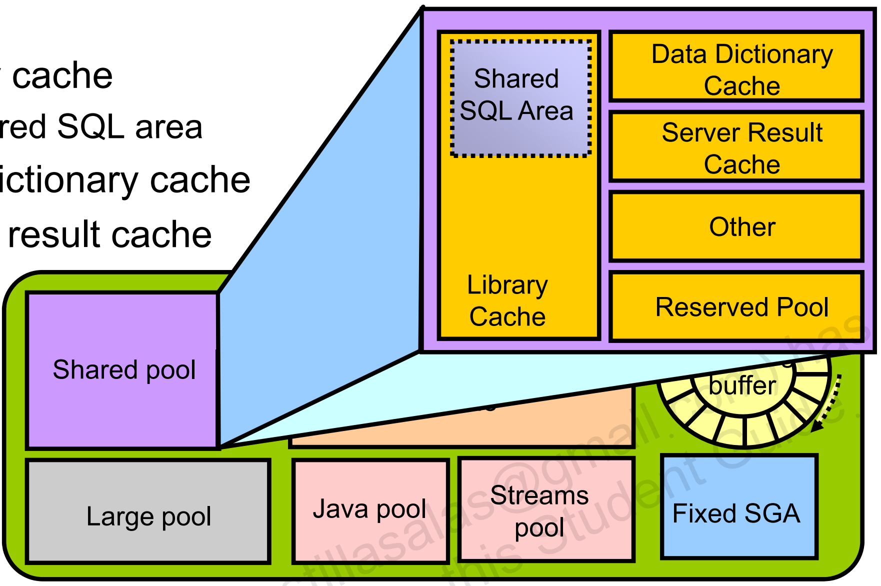
A Program Global Area (PGA) is a memory region that contains data and control information for each server process. An Oracle server process services a client's requests. Each server process has its own private PGA that is allocated when the server process is started. Access to the PGA is exclusive to that server process, and the PGA is read and written only by the Oracle code acting on its behalf. The PGA is divided into two major areas: stack space and the user global area (UGA).

With the dynamic SGA infrastructure, the sizes of the database buffer cache, the shared pool, the large pool, the Java pool, and the Streams pool can change without shutting down the instance.

The Oracle Database server uses initialization parameters to create and manage memory structures. The simplest way to manage memory is to allow the database to automatically manage and tune it for you. To do so (on most platforms), you only have to set a target memory size initialization parameter (`MEMORY_TARGET`) and a maximum memory size initialization parameter (`MEMORY_MAX_TARGET`).

Shared Pool

- Is a portion of the SGA
- Contains:
  - Library cache
    - Shared SQL area
  - Data dictionary cache
  - Server result cache



System Global Area (SGA)

The shared pool portion of the SGA contains the library cache, the data dictionary cache, the server result cache containing the SQL query result cache and the PL/SQL function result cache, buffers for parallel execution messages, and control structures.

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. This access is essential to the continuing operation of Oracle Database.

The data dictionary is accessed so often by Oracle Database that two special locations in memory are designated to hold dictionary data. One area is called the *data dictionary cache*, also known as the row cache because it holds data as rows instead of buffers (buffers hold entire blocks of data). The other area in memory that holds dictionary data is the *library cache*. All Oracle Database user processes share these two caches for access to data dictionary information.

Oracle Database represents each SQL statement that it runs with a shared SQL area (as well as a private SQL area kept in the PGA). Oracle Database recognizes when two users are executing the same SQL statement and reuses the shared SQL area for those users.

A shared SQL area contains the parse tree and execution plan for a given SQL statement. Oracle Database saves memory by using one shared SQL area for SQL statements run multiple times, which often happens when many users run the same application.

When a new SQL statement is parsed, Oracle Database allocates memory from the shared pool to store in the shared SQL area. The size of this memory depends on the complexity of the statement.

Oracle Database processes PL/SQL program units (procedures, functions, packages, anonymous blocks, and database triggers) in much the same way it processes individual SQL statements. Oracle Database allocates a shared area to hold the parsed, compiled form of a program unit. Oracle Database allocates a private area to hold values specific to the session that runs the program unit, including local, global, and package variables (also known as package instantiation) and buffers for executing SQL. If more than one user runs the same program unit, then a single, shared area is used by all users, while all users maintain separate copies of their own private SQL areas, holding values specific to their own sessions.

Individual SQL statements contained in a PL/SQL program unit are processed just like other SQL statements. Despite their origins in 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.

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

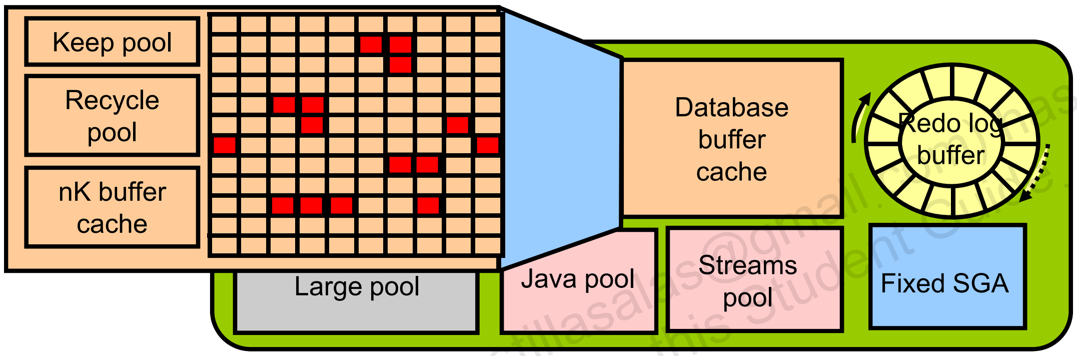
Results of queries and query fragments can be cached in memory in the SQL query result cache. The database server can then use cached results to answer future executions of these queries and query fragments. Because retrieving results from the SQL query result cache is faster than rerunning a query, frequently run queries experience a significant performance improvement when their results are cached.

A PL/SQL function is sometimes used to return the result of a computation whose inputs are one or several parameterized queries issued by the function. In some cases, these queries access data that changes very infrequently compared to the frequency of calling the function. You can include syntax in the source text of a PL/SQL function to request that its results be cached in the PL/SQL function result cache and (to ensure correctness) that the cache be purged when tables in a list of tables experience data manipulation language (DML).

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

Database Buffer Cache

- Is part of the SGA
- Holds copies of data blocks that are read from data files
- Is shared by all concurrent users



System Global Area (SGA)

The database buffer cache is the portion of the SGA that holds block images read from the data files or constructed dynamically to satisfy the read consistency model. All users who are concurrently connected to the instance share access to the database buffer cache.

The first time an Oracle Database user process requires a particular piece of data, it searches for the data in the database buffer cache. If the process finds the data already in the cache (a cache hit), it can read the data directly from memory. If the process cannot find the data in the cache (a cache miss), it must copy the data block from a data file on disk into a buffer in the cache before accessing the data. Accessing data through a cache hit is faster than accessing data through a cache miss.

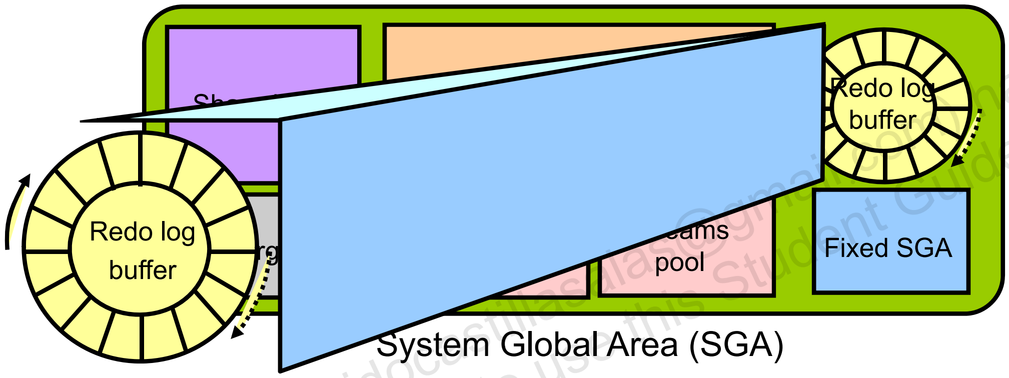
The buffers in the cache are managed by a complex algorithm that uses a combination of least recently used (LRU) lists and touch count. The LRU helps to ensure that the most recently used blocks tend to stay in memory to minimize disk access.

The keep buffer pool and the recycle buffer pool are used for specialized buffer pool tuning. The keep buffer pool is designed to retain buffers in memory longer than the LRU would normally retain them. The recycle buffer pool is designed to flush buffers from memory faster than the LRU normally would.

Additional buffer caches can be configured to hold blocks of a size that is different from the default block size.

Redo Log Buffer

- Is a circular buffer in the SGA
- Holds information about changes made to the database
- Contains redo entries that have the information to redo changes made by operations such as DML and DDL



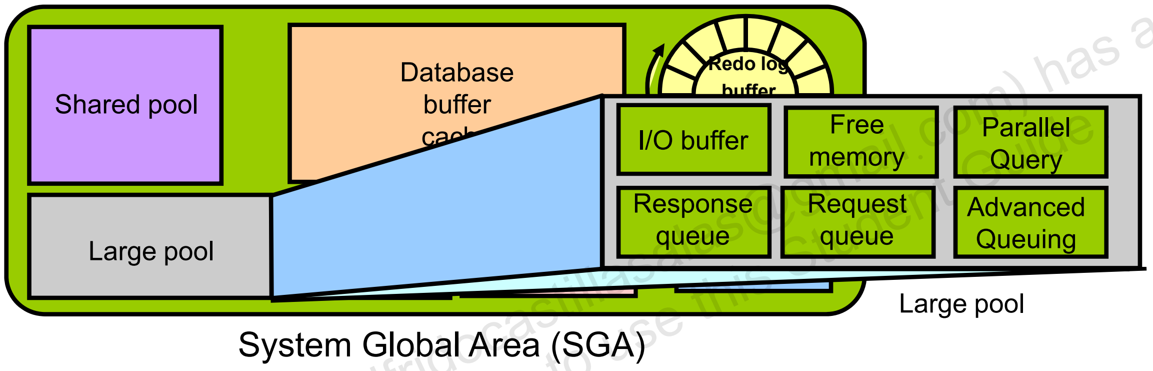
The redo log buffer is a circular buffer in the SGA that holds information about changes made to the database. This information is stored in redo entries. Redo entries contain the information necessary to reconstruct (or redo) changes that are made to the database by DML, DDL, or internal operations. Redo entries are used for database recovery if necessary.

As the server process makes changes to the buffer cache, redo entries are generated and written to the redo log buffer in the SGA. The redo entries take up continuous, sequential space in the buffer. The log writer background process writes the redo log buffer to the active redo log file (or group of files) on disk.

Large Pool

Provides large memory allocations for:

- Session memory for the shared server and the Oracle XA interface
- I/O server processes
- Oracle Database backup and restore operations



The database administrator can configure an optional memory area called the *large pool* to provide large memory allocations for:

- Session memory for the shared server and the Oracle XA interface (used where transactions interact with multiple databases)
- I/O server processes
- Oracle Database backup and restore operations
- Parallel Query operations
- Advanced Queuing memory table storage

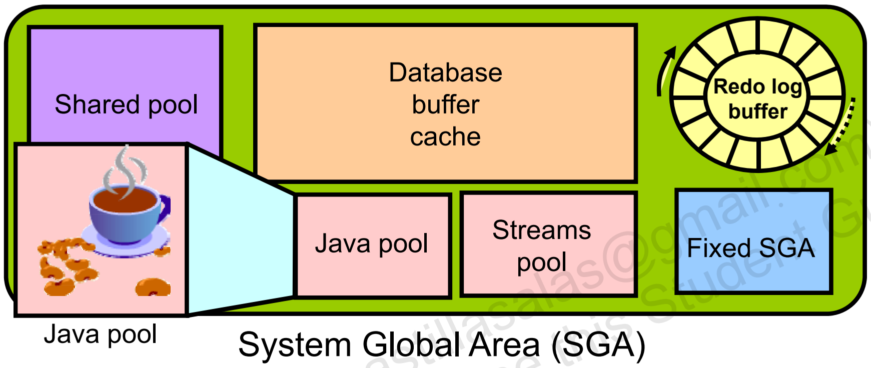
By allocating session memory from the large pool for shared server, Oracle XA, or parallel query buffers, Oracle Database can use the shared pool primarily for caching shared SQL and avoid the performance overhead that is caused by shrinking the shared SQL cache.

In addition, the memory for Oracle Database backup and restore operations, for I/O server processes, and for parallel buffers is allocated in buffers of a few hundred kilobytes. The large pool is better able to satisfy such large memory requests than the shared pool.

The large pool is not managed by a least recently used (LRU) list.

Java Pool

Java pool memory is used to store all session-specific Java code and data in the JVM.

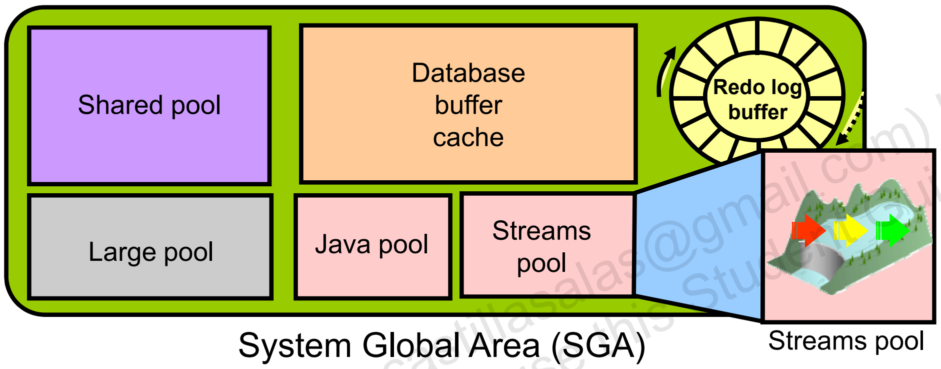


Java pool memory is used to store all session-specific Java code and data in the Java Virtual Machine (JVM). Java pool memory is used in different ways, depending on the mode in which Oracle Database is running.

Streams Pool

Streams pool memory is used exclusively by Oracle Streams to:

- Store buffered queue messages
- Provide memory for Oracle Streams processes

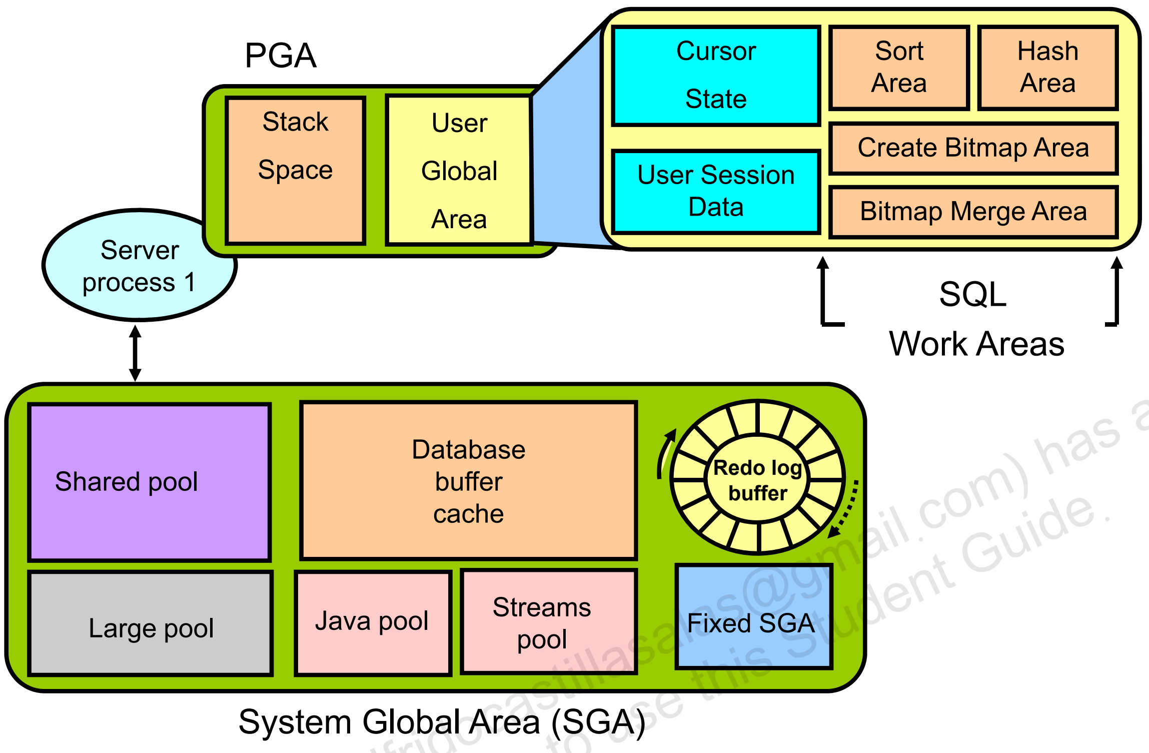


System Global Area (SGA)

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

Unless you specifically configure it, the size of the Streams pool starts at zero. The pool size grows dynamically as needed when Oracle Streams is used.

Program Global Area (PGA)



The Program Global Area (PGA) is a private memory region containing data and control information for a server process. Each server process has a distinct PGA. Access to it is exclusive to that server process and it is read only by Oracle code acting on behalf of it. It is not available for developer's code.

Every PGA contains stack space. In a dedicated server environment, each user connecting to the database instance has a separate server process. For this type of connection, the PGA contains a subdivision of memory known as the user global area (UGA). The UGA is composed of the following:

- Cursor area for storing runtime information on cursors
- User session data storage area for control information about a session
- SQL working areas for processing SQL statements consisting of:
  - A sort area for functions that order data such as `ORDER BY` and `GROUP BY`
  - A hash area for performing hash joins of tables
  - A create bitmap area used in bitmap index creation common to data warehouses
  - A bitmap merge area used for resolving bitmap index plan execution

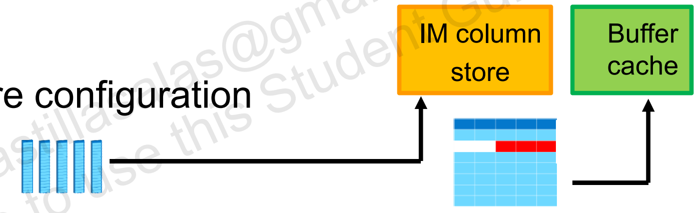
In a shared server environment, multiple client users share the server process. In this model, the UGA is moved into the SGA (shared pool or large pool if configured) leaving the PGA with only stack space.

12.1.0.2

In-Memory Column Store: Introduction

- Instant query response:
  - Faster queries on very large tables on any columns (100x)
  - Use of scans, joins, and aggregates
  - Without indexes
  - Best suited for analytics: few columns, many rows
- Faster DML: Removal of most analytics indexes (3 to 4x)
- Full application transparency
- Easy setup:
  - In-memory column store configuration
  - Segment attributes

ORACLE E-BUSINESS SUITE, ORACLE FUSION APPLICATIONS, ORACLE SIEBEL, ORACLE JD EDWARDS, ORACLE PEOPLESOFT



The In-Memory Column Store feature enables objects (tables, partitions, and other types) to be stored in memory in a new format known as the *columnar format*. This format enables scans, joins, and aggregates to perform much faster than the traditional on-disk format, thus providing fast reporting and DML performance for both OLTP and DW environments. This is particularly useful for analytic applications that operate on few columns returning many rows rather than for OLTP that operates on few rows returning many columns. The DBA must define the segments that are to be populated into the in-memory column store (IM column store), such as hot tables, partitions, and more precisely the more frequently accessed columns.

The in-memory columnar format does not replace the on-disk or buffer cache format. It is a consistent copy of a table or of some columns of a table converted to the new columnar format that is independent of the disk format and only available in memory. Because of this independence, applications are able to transparently use this option without any changes. For the data to be converted into the new columnar format, a new pool is requested in the SGA. The pool is the IM column store.

If sufficient space is allocated for the IM column store, a query that accesses objects that are candidates to be populated into the IM column store performs much faster. The improved performance allows ad hoc analytic queries to be executed directly on the real-time transaction data without impacting the existing workload.

There are three main advantages:

- Queries run a lot faster: All data can be populated in memory in a compressed columnar format. No index is required and used. Queries run at least 100 times faster than when fetching data from the buffer cache, thanks to the columnar compressed format.
- DMLs are faster: Analytics indexes can be eliminated by being replaced by scans of the IM column store representation of the table.
- Arbitrary ad hoc queries run with good performance, because the table behaves as if all columns are indexed.

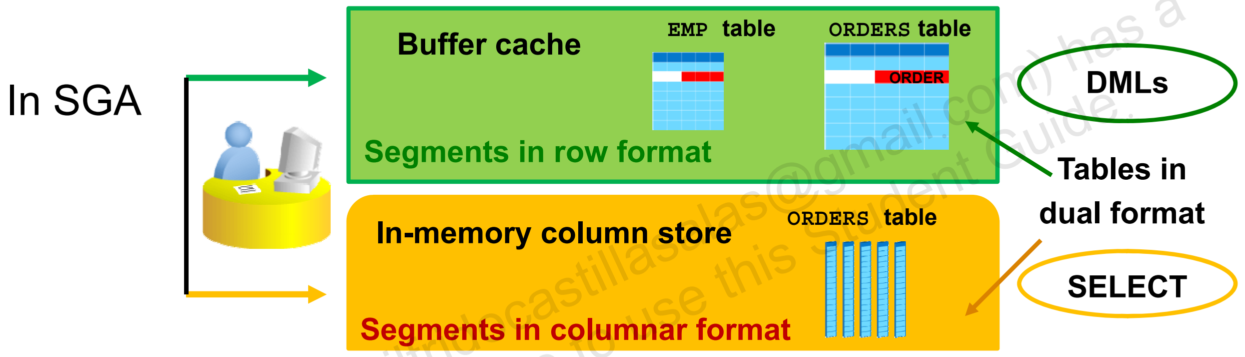
Note: The In-Memory Column Store feature is included with the Oracle Database In-Memory option.

Refer to the *Oracle Database Administrator's Guide* for detailed information about this feature.

12.1.0.2

In-Memory Column Store: Overview

- A new pool in the SGA: In-Memory column store
  - Segments populated into the IM column store are converted into a columnar format.
  - In-Memory segments are transactionally consistent with the buffer cache.
- Only one segment on disk and in row format



The in-memory columnar format does not replace the on-disk or buffer cache format. This means that when a segment such as a table or a partition is populated into the IM column store, the on-disk format segment is automatically converted into a columnar format and optionally compressed. The columnar format is a pure in-memory format. There is no columnar format storage on disk. It never causes additional writes to disk and therefore does not require any logging or undo space.

All data is stored on disk in the traditional row format.

Moreover, the columnar format of a segment is a transaction-consistent copy of the segment either on disk or in the buffer cache. Transaction consistency between the two pools is maintained.

If sufficient space is allocated to the IM column store in SGA, a query that accesses objects that are populated into the IM column store performs much faster. The improved performance allows more ad hoc analytic queries to be executed directly on real-time transaction data without impacting the existing workload. A lack of IM column store space does not prevent statements from executing against tables that could have been populated into the IM column store.

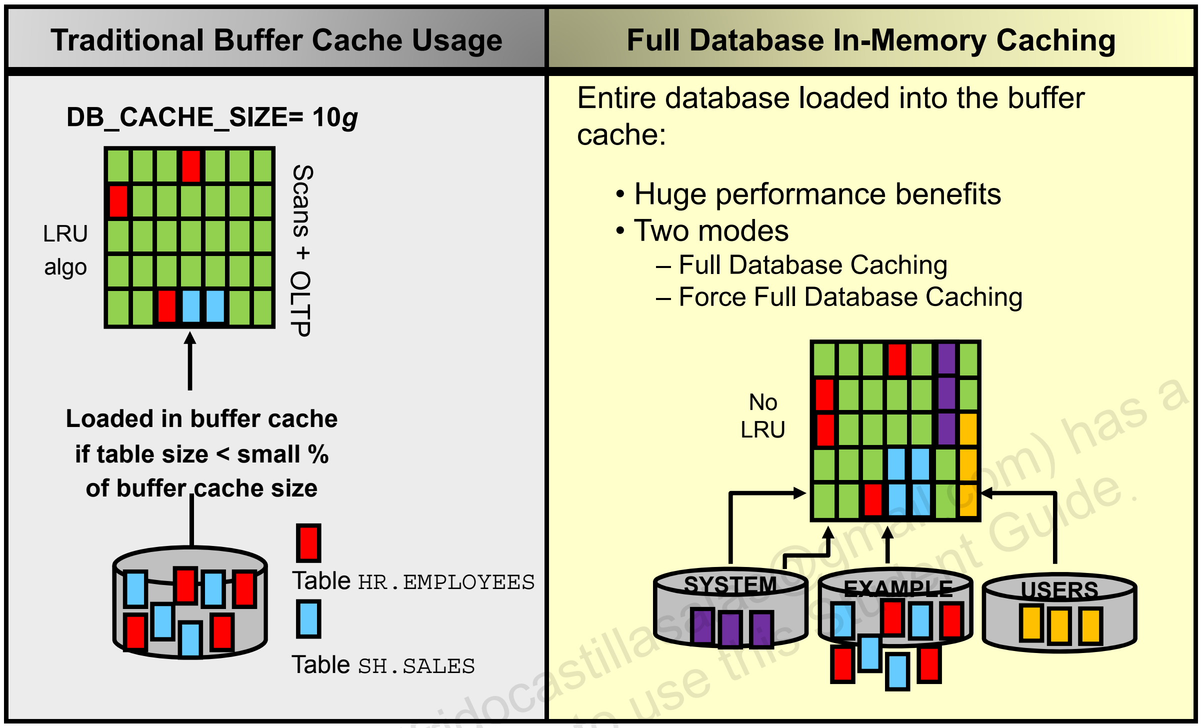
The DBA must decide, according to the types of queries and DMLs that are executed against the segments, which segments should be defined as non in-memory segments and which should be defined as in-memory segments. The DBA can also define more precisely which columns are good candidates for IM column store:

- In **row format exclusively:** The segments that are being frequently accessed by OLTP-style queries, which operate on few rows returning many columns, are good candidates for the buffer cache. These segments should not necessarily be defined as in-memory segments and should be sent to the buffer cache only.
- In **dual format simultaneously:** The segments that are being frequently accessed by analytical-style queries, which operate on many rows returning a few columns, are good candidates for IM column store. If a segment is defined as an in-memory segment but has some columns that are defined as non in-memory columns, the queries that select any non in-memory columns are sent to the buffer cache and those selecting in-memory columns only are sent to the IM column store. Any fetch-by-rowid is performed on the segment through the buffer cache.

Any DML performed on these objects is executed via the buffer cache.

12.1.0.2

Full Database In-Memory Caching


Traditional Buffer Cache Usage
Full Database In-Memory Caching
DB_CACHE_SIZE= 10g
LRU algo
Scans + OLTP
Loaded in buffer cache if table size < small % of buffer cache size
Table HR.EMPLOYEES
Table SH.SALES
Entire database loaded into the buffer cache:
• Huge performance benefits
• Two modes
– Full Database Caching
– Force Full Database Caching
No LRU
SYSTEM
EXAMPLE
USERS

The current algorithm for table scans loads a table into the buffer cache only when the table size is less than a small percent of the buffer cache size. For very large tables, the database uses a direct path read, which loads blocks directly into the PGA and bypasses the SGA, to avoid flooding the buffer cache. The DBA must explicitly declare small lookup tables, which are accessed frequently, as `CACHE` to load data into memory and avoid bypassing the SGA. This clause indicates that the blocks retrieved for these tables are placed at the most recently used end of the least recently used (LRU) list in the buffer cache when a full table scan is performed.

The Full Database In-memory Caching feature enables an entire database to be cached in memory when the database size (sum of all data files, `SYSTEM` tablespace, LOB CACHE files minus `SYSAUX`, `TEMP`) is smaller than the buffer cache size. Caching and running a database from memory leads to huge performance benefits. Two modes can be used:

- **Full Database Caching:** Implicit default and automatic mode in which an internal calculation determines if the database can be fully cached for an instance. `NOCACHE` LOBs are not cached in Full Database Caching but in Force Full Database Caching mode even `NOCACHE` LOBs are cached.

- **Force Full Database Caching:** Neither Full Database Caching nor Force Full Database Caching forces or prefetches data into memory. Workload must access the data first for them to be cached. It considers the entire database as eligible to be completely cached in the buffer cache. This mode requires the DBA to execute the `ALTER DATABASE FORCE FULL DATABASE CACHING` command. This mode takes precedence over Full Database Caching mode. To revert to traditional caching, use the `ALTER DATABASE NO FORCE FULL DATABASE CACHING` command.

Refer to the *Oracle Database Administrators Guide* and the *Oracle Database Performance Tuning Guide* for detailed information about this feature.

Quiz

The memory region that contains data and control information for a server or background process is called:

a. Shared pool
b. PGA
c. Buffer cache
d. User session data

Answer: b

Quiz

What is read into the database buffer cache from data files?

a. Rows
b. Changes
c. Blocks
d. SQL

Answer: c

Process Architecture

- User process
  - Is the application or tool that connects to the Oracle database
- Database processes
  - Server process: Connects to the Oracle instance and is started when a user establishes a session
  - Background processes: Are started when an Oracle instance is started
- Daemon / Application processes
  - Networking listeners
  - Grid Infrastructure daemons

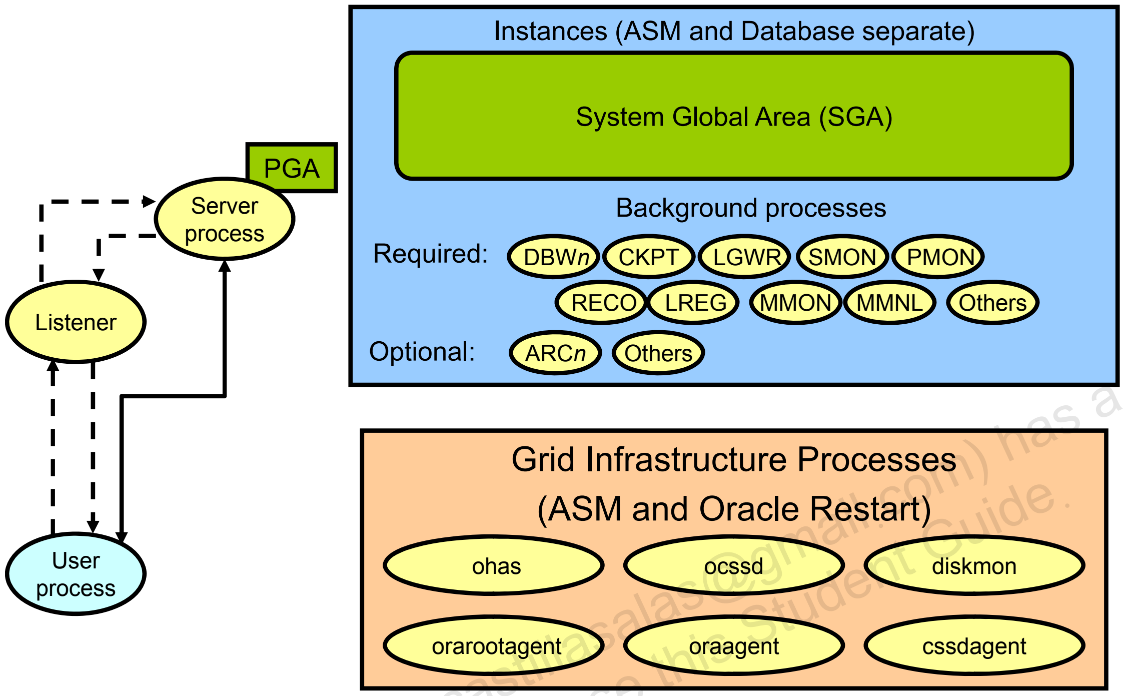
The processes in an Oracle Database system can be divided into three major groups:

- User processes that run the application or Oracle tool code
- Oracle Database processes that run the Oracle Database server code (including server processes and background processes)
- Oracle daemons and application processes not specific to a single database

When a user runs an application program or an Oracle tool such as SQL*Plus, the term *user process* is used to refer to the user's application. The user process may or may not be on the database server machine. Oracle Database also creates a *server process* to execute the commands issued by the user process. In addition, the Oracle server also has a set of *background processes* for an instance that interact with each other and with the operating system to manage the memory structures, asynchronously perform I/O to write data to disk, and perform other required tasks. The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options. The code for connected users can be configured as a dedicated server or a shared server.

- **Dedicated server:** For each session, the database application is run by a user process that is served by a dedicated server process that executes Oracle database server code.
- **Shared server:** Eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. A shared server process serves any client request.

Process Structures



Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to the instance. The user process represents the application or tool that connects to the Oracle database. It may be on the same machine as the Oracle database, or it may exist on a remote client and use a network to reach the Oracle database. The user process first communicates with a listener process that creates a server process in a dedicated environment.

Server processes created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application.
- Read necessary data blocks from data files on disk into the shared database buffers of the SGA (if the blocks are not already present in the SGA).
- Return results in such a way that the application can process the information.

Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called *background processes*. An Oracle Database instance can have many background processes.

The background processes commonly seen in non-RAC, non-ASM environments can include the following:

- Database Writer process (DBW*n*)
- Log Writer process (LGWR)
- Checkpoint process (CKPT)
- System monitor process (SMON)
- Process monitor process (PMON)
- Recoverer process (RECO)
- Listener registration process (LREG)
- Manageability monitor process (MMON)
- Manageability monitor lite process (MMNL)
- Job queue coordinator (CJQ0)
- Job slave processes (J*nnn*)
- Archiver processes (ARC*n*)
- Queue monitor processes (QMN*n*)

Other background processes may be found in more advanced configurations such as RAC. See the `V$BGPROCESS` view for more information on the background processes.

Some background processes are created automatically when an instance is started, whereas others are started as required.

Other process structures are not specific to a single database, but rather can be shared among many databases on the same server. The Grid Infrastructure and networking processes fall into this category.

Oracle Grid Infrastructure processes on Linux and UNIX systems include the following:

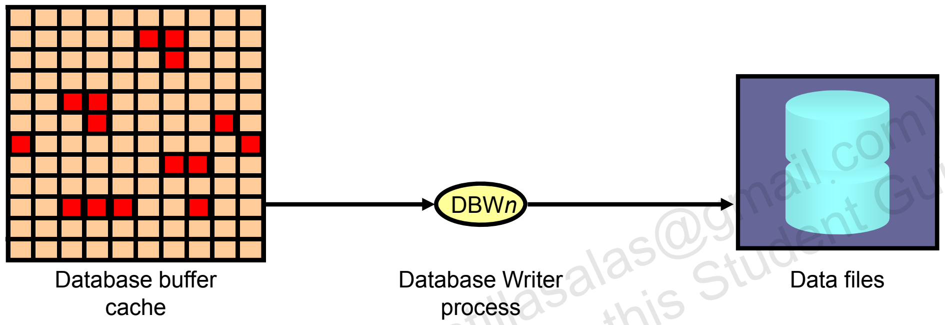
- `ohasd` (Oracle High Availability Service daemon): Is responsible for starting Oracle Clusterware processes
- `ocssd`: Cluster Synchronization Service daemon
- `diskmon` (Disk Monitor daemon): Is responsible for input and output fencing for Oracle Exadata Storage
- `cssdagent`: Starts, stops, and checks the status of the CSS daemon, `ocssd`
- `oraagent`: Extends Clusterware to support Oracle-specific requirements and complex resources
- `orarootagent`: Is a specialized Oracle agent process that helps manage resources owned by root, such as the network.

Note: For a more detailed list of the background processes, consult the *Oracle Database Reference* guide.

Database Writer Process (DBWn)

Writes modified (dirty) buffers in the database buffer cache to disk:

- Asynchronously while performing other processing
- To advance the checkpoint



The Database Writer process (DBW*n*) writes the contents of buffers to data files. The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one Database Writer process (DBW*0*) is adequate for most systems, you can configure additional processes to improve write performance if your system modifies data heavily. The additional processes are named DBW1 through DBW9, DBWa through DBWz, and BW36-BW99.These additional DBW*n* processes are not useful on uniprocessor systems.

When a buffer in the database buffer cache is modified, it is marked dirty and is added to the head of the checkpoint queue that is kept in system change number (SCN) order. This order therefore matches the order of redo that is written to the redo logs for these changed buffers. When the number of available buffers in the buffer cache falls below an internal threshold (to the extent that server processes find it difficult to obtain available buffers), DBW*n* writes non-frequently used buffers to the data files from the tail of the LRU list so that processes can replace buffers when they need them. DBW*n* also writes from the tail of the checkpoint queue to keep the checkpoint advancing.

The SGA contains a memory structure that has the redo byte address (RBA) of the position in the redo stream where recovery should begin in case of an instance failure. This structure acts as a pointer into the redo and is written to the control file by the CKPT process once every three seconds. Because the DBW*n* writes dirty buffers in SCN order, and because the redo is in SCN order, every time DBW*n* writes dirty buffers from the LRU list, it also advances the pointer held in the SGA memory structure so that instance recovery (if required) begins reading the redo from approximately the correct location and avoids unnecessary I/O. This is known as *incremental checkpointing*.

Note: There are other cases when DBW*n* may write (for example, when tablespaces are made read-only or are placed offline). In such cases, no incremental checkpoint occurs because dirty buffers belonging only to the corresponding data files are written to the database unrelated to the SCN order.

The LRU algorithm keeps more frequently accessed blocks in the buffer cache to minimize disk reads. A `CACHE` option can be placed on tables to help retain blocks even longer in memory.

The `DB_WRITER_PROCESSES` initialization parameter specifies the number of DBW*n* processes. The maximum number of Database Writer processes is 100. If it is not specified by the user during startup, Oracle Database determines how to set `DB_WRITER_PROCESSES` based on the number of CPUs and processor groups.

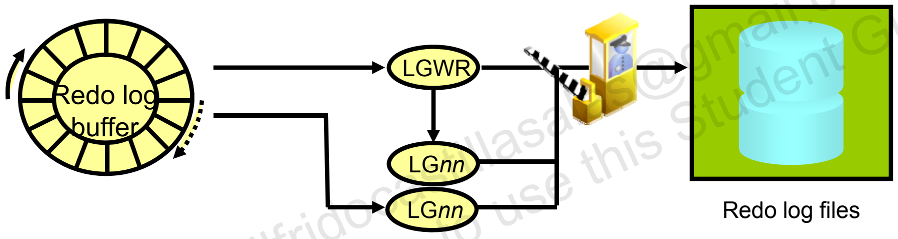
The DBW*n* process writes dirty buffers to disk under the following conditions:

- When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers, it signals DBW*n* to write. DBW*n* writes dirty buffers to disk asynchronously while performing other processing.
- DBW*n* writes buffers to advance the checkpoint, which is the position in the redo thread (log) from which instance recovery begins. This log position is determined by the oldest dirty buffer in the buffer cache.

In all cases, DBW*n* performs batched (multiblock) writes to improve efficiency. The number of blocks written in a multiblock write varies by operating system.

Log Writer Process (LGWR)

- Writes the redo log buffer to a redo log file on disk
  - When a user process commits a transaction
  - When an online redo log switch occurs
  - When the redo log buffer is one-third full or contains 1 MB of buffered data
  - Before a DBW*n* process writes modified buffers to disk
  - When three seconds have passed since the last write
- Serves as coordinator of LG*nn* processes and ensures correct order for operations that must be ordered



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

LGWR starts and coordinates multiple helper processes that concurrently perform some of the work. LGWR handles the operations that are very fast, or must be coordinated, and delegates operations to the LG*nn* that could benefit from concurrent operations, primarily writing the redo from the log buffer to the redo log file and posting the completed write to the foreground process that is waiting.

Because LG*nn* processes work concurrently and certain operations must be performed in order, LGWR forces ordering so that even if the writes complete out of order, the posting to the foreground processes will be in the correct order.

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

LGWR writes:

- When a user process commits a transaction
- When an online redo log switch occurs
- When the redo log buffer is one-third full or contains 1 MB of buffered data
- Before a DBWn process writes modified buffers to disk (if necessary)
- When three seconds have passed since the last write to log files

Before DBW*n* can write a modified buffer, all redo records that are associated with the changes to the buffer must be written to disk (the write-ahead protocol). If DBW*n* finds that some redo records have not been written, it signals LGWR to write the redo records to disk and waits for LGWR to complete writing the redo log buffer before it can write out the data buffers. LGWR writes to the current log group. If one of the files in the group is damaged or unavailable, LGWR continues writing to other files in the group and logs an error in the LGWR trace file and in the system alert log. If all files in a group are damaged, or if the group is unavailable because it has not been archived, LGWR cannot continue to function.

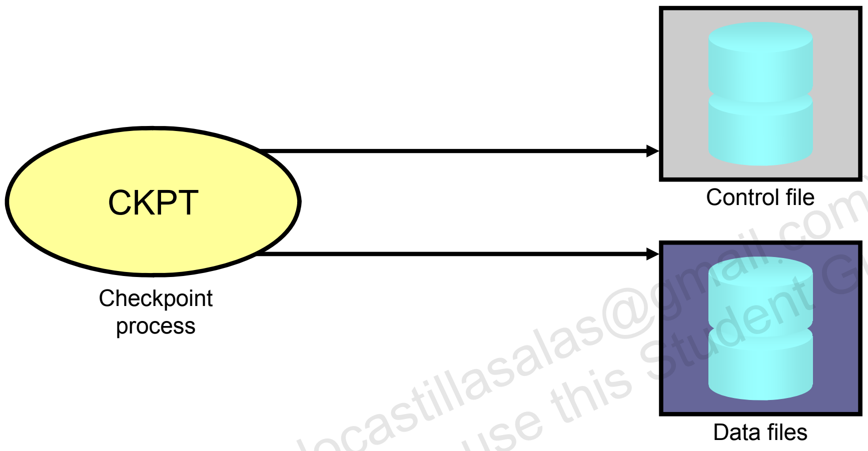
When a user issues a `COMMIT` statement, LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries. The corresponding changes to data blocks are deferred until it is more efficient to write them. This is called a *fast commit mechanism*. The atomic write of the redo entry containing the transaction's commit record is the single event that determines whether the transaction has committed. Oracle Database returns a success code to the committing transaction, although the data buffers have not yet been written to disk.

If more buffer space is needed, LGWR sometimes writes redo log entries before a transaction is committed. These entries become permanent only if the transaction is later committed. When a user commits a transaction, the transaction is assigned an SCN, which Oracle Database records along with the transaction's redo entries in the redo log. SCNs are recorded in the redo log so that recovery operations can be synchronized in Real Application Clusters and distributed databases.

In times of high activity, LGWR can write to the redo log file by using group commits. For example, suppose that a user commits a transaction. LGWR must write the transaction's redo entries to disk. As this happens, other users issue `COMMIT` statements. However, LGWR cannot write to the redo log file to commit these transactions until it has completed its previous write operation. After the first transaction's entries are written to the redo log file, the entire list of redo entries of waiting transactions (not yet committed) can be written to disk in one operation, requiring less I/O than do transaction entries handled individually. Therefore, Oracle Database minimizes disk I/O and maximizes performance of LGWR. If requests to commit continue at a high rate, every write (by LGWR) from the redo log buffer can contain multiple commit records.

Checkpoint Process (CKPT)

- Records checkpoint information in
  - Control file
  - Each data file header
- Signals DBW*n* to write blocks to disk

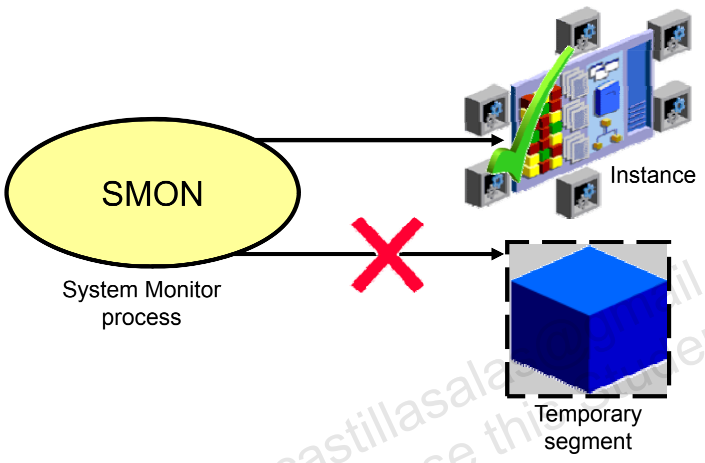


A *checkpoint* is a data structure that defines a system change number (SCN) in the redo thread of a database. Checkpoints are recorded in the control file and in each data file header. They are a crucial element of recovery.

When a checkpoint occurs, Oracle Database must update the headers of all data files to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBW*n* always performs that work. The SCNs recorded in the file headers guarantee that all changes made to database blocks before that SCN have been written to disk.

System Monitor Process (SMON)

- Performs recovery at instance startup
- Cleans up unused temporary segments

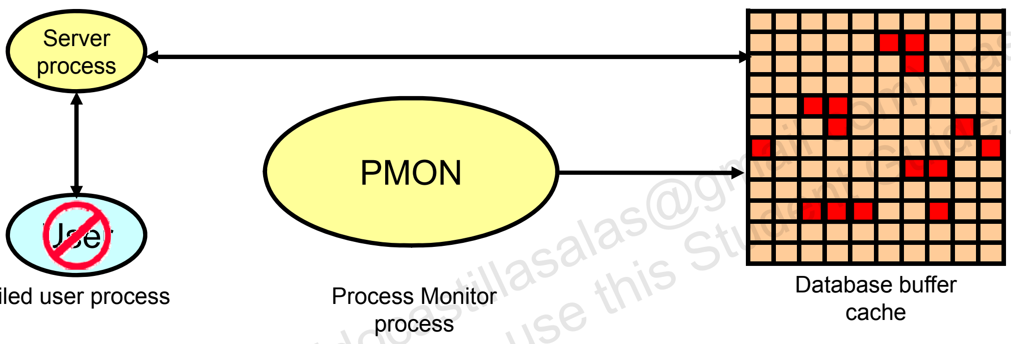


The System Monitor process (SMON) performs recovery at instance startup if necessary. SMON is also responsible for cleaning up temporary segments that are no longer in use. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online.

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

Process Monitor Process (PMON)

- Performs process recovery when a user process fails
  - Cleans up the database buffer cache
  - Frees resources that are used by the user process
- Monitors sessions for idle session timeout



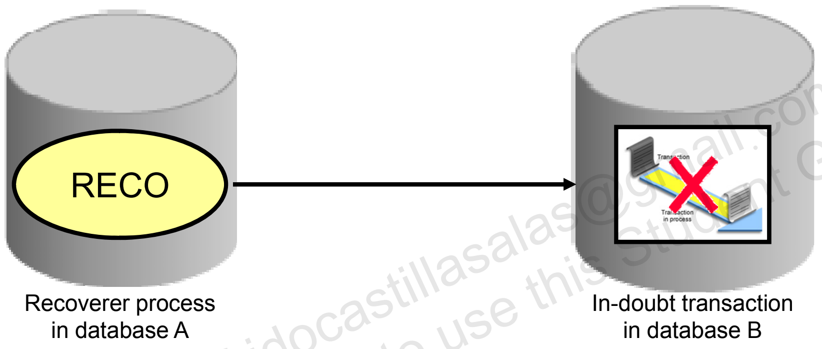
The Process Monitor process (PMON) performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.

PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle Database has terminated intentionally).

Like SMON, PMON checks regularly to see whether it is needed; it can be called if another process detects the need for it.

Recoverer Process (RECO)

- Used with the distributed database configuration
- Automatically connects to other databases involved in in-doubt distributed transactions
- Automatically resolves all in-doubt transactions
- Removes any rows that correspond to in-doubt transactions

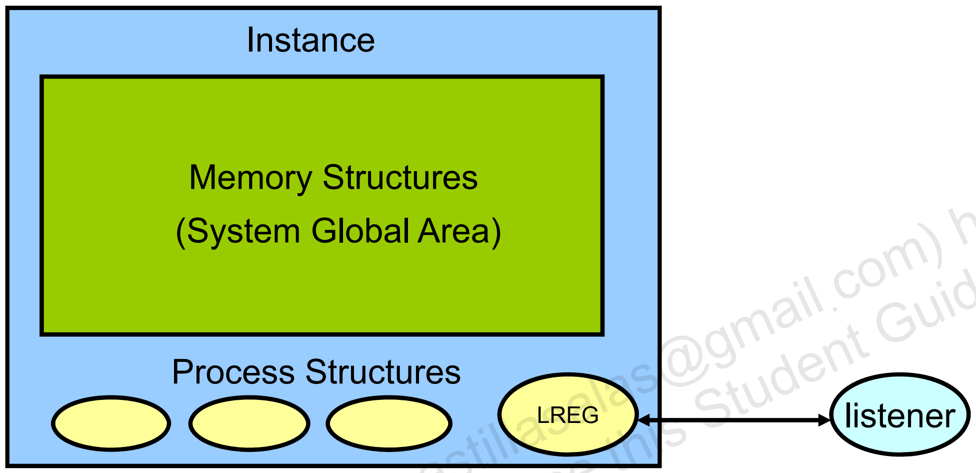

RECO
Recoverer process in database A
In-doubt transaction in database B

The Recoverer process (RECO) is a background process that is used with the distributed database configuration that automatically resolves failures involving distributed transactions. The RECO process of an instance automatically connects to other databases involved in an in-doubt distributed transaction. When the RECO process re-establishes a connection between involved database servers, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved in-doubt transactions.

If the RECO process fails to connect with a remote server, RECO automatically tries to connect again after a timed interval. However, RECO waits an increasing amount of time (growing exponentially) before it attempts another connection.

Listener Registration Process (LREG)

Registers information about the database instance and dispatcher processes with the Oracle Net Listener



The Listener Registration process, LREG, registers information about the database instance and dispatcher processes with the Oracle Net Listener. LREG provides the listener with the following information:

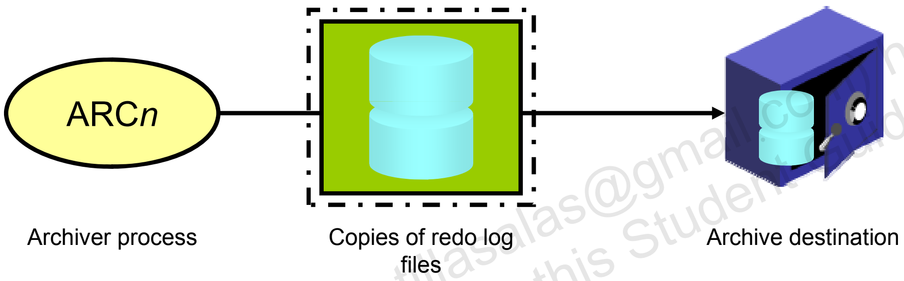
- Names of the database services
- Name of the database instance associated with the services and its current and maximum load
- Service handlers (dispatchers and dedicated servers) available for the instance, including their type, protocol addresses, and current and maximum load

When the instance starts, LREG attempts to connect to the listener. If the listener is running, LREG passes information to it. If the listener is not running, LREG periodically attempts to connect to it. It may take up to 60 seconds for LREG to register the database instance with the listener after the listener has started.

You can use the `ALTER SYSTEM REGISTER` command to immediately initiate service registration after starting the listener.

Archiver Processes (ARC*n*)

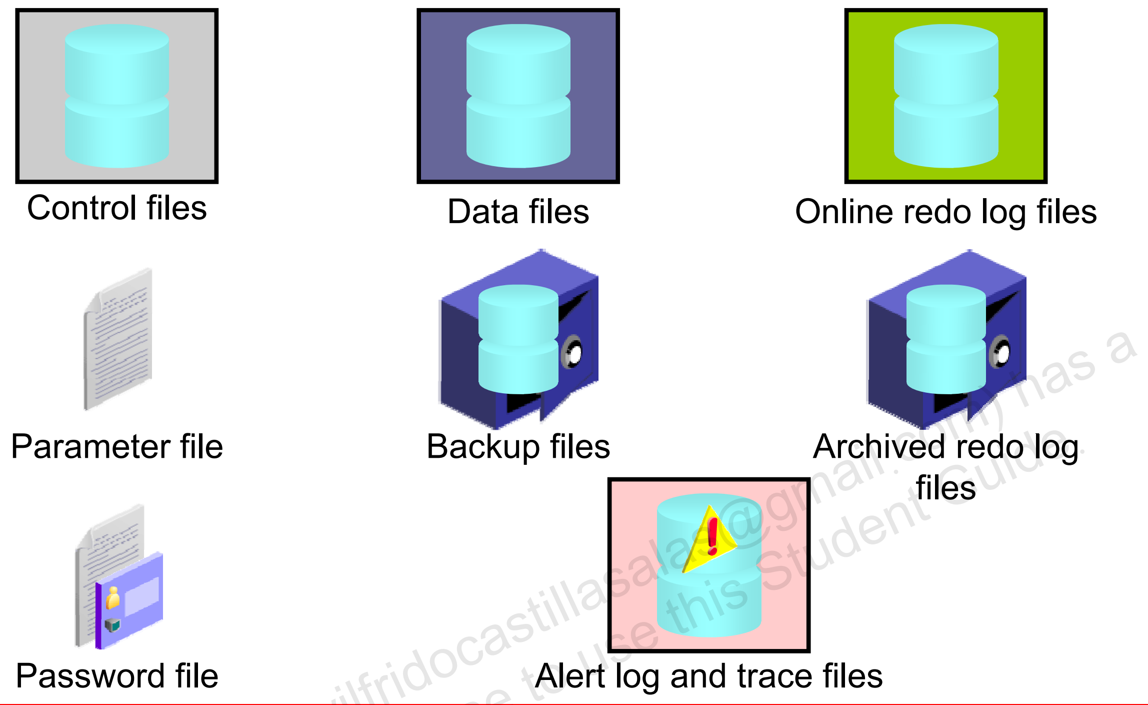
- Copy redo log files to a designated storage device after a log switch has occurred
- Can collect transaction redo data and transmit that data to standby destinations



The Archiver processes (ARC*n*) copy redo log files to a designated storage device after a log switch has occurred. ARC*n* processes are present only when the database is in `ARCHIVELOG` mode and automatic archiving is enabled.

If you anticipate a heavy workload for archiving (such as during bulk loading of data), you can increase the maximum number of Archiver processes. There can also be multiple archive log destinations. It is recommended that there be at least one Archiver process for each destination. The default is to have four Archiver processes.

Database Storage Architecture



The files that comprise an Oracle database are as follows:

- **Control files:** Each database has one unique control file that contains data about the database itself (that is, physical database structure information). Multiple copies may be maintained to protect against total loss. It can also contain metadata related to backups. The control file is critical to the database. Without the control file, the database cannot be opened.
- **Data files:** Contain the user or application data of the database, as well as metadata and the data dictionary
- **Online redo log files:** Allow for instance recovery of the database. If the database server crashes and does not lose any data files, the instance can recover the database with the information in these files.

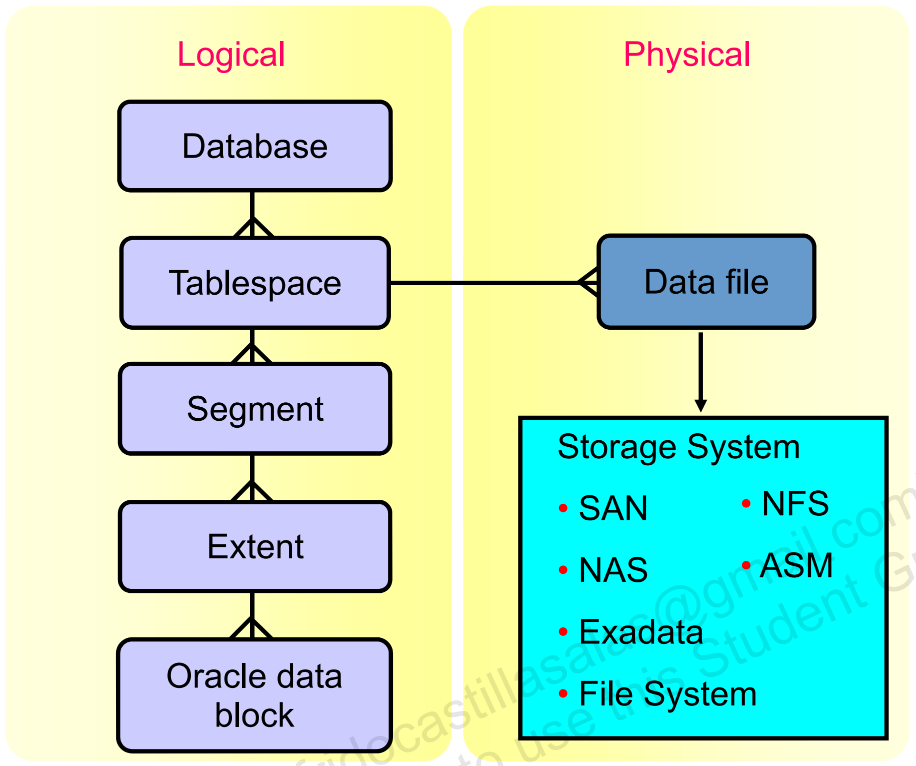
The following additional files are used during the operation of the database:

- **Parameter file:** Is used to define how the instance is configured when it starts up
- **Password file:** Allows users using the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, `SYSKM`, and `SYSASM` roles to connect remotely to the instance and perform administrative tasks
- **Backup files:** Are used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.

- **Archived redo log files:** Contain an ongoing history of the data changes (redo) that are generated by the instance. Using these files and a backup of the database, you can recover a lost data file. That is, archive logs enable the recovery of restored data files.
- **Trace files:** Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, whereas other information is for Oracle Support Services.
- **Alert log file:** These are special trace entries. The alert log of a database is a chronological log of messages and errors. Oracle recommends that you review the alert log periodically.

Note: Parameter, password, alert, and trace files are covered in other lessons.

Logical and Physical Database Structures



The database has logical structures and physical structures.

Databases, Tablespaces, and Data Files

The relationship among databases, tablespaces, and data files is illustrated in the slide. Each database is logically divided into two or more tablespaces. One or more data files are explicitly created for each tablespace to physically store the data of all segments in a tablespace. If it is a `TEMPORARY` tablespace, it has a temporary file instead of a data file. A tablespace's data file can be physically stored on any supported storage technology.

Tablespaces

A database is divided into logical storage units called *tablespaces*, which group related logical structures or data files together. For example, tablespaces commonly group all of an application's segments to simplify some administrative operations.

Data Blocks

At the finest level of granularity, an Oracle database's data is stored in *data blocks*. One data block corresponds to a specific number of bytes of physical space on the disk. A data block size is specified for each tablespace when it is created. A database uses and allocates free database space in Oracle data blocks.

Extents

The next level of logical database space is an *extent*. An extent is a specific number of contiguous Oracle data blocks (obtained in a single allocation) that are used to store a specific type of information. Oracle data blocks in an extent are logically contiguous but can be physically spread out on disk because of RAID striping and file system implementations.

Segments

The level of logical database storage above an extent is called a *segment*. A segment is a set of extents allocated for a certain logical structure. Example:

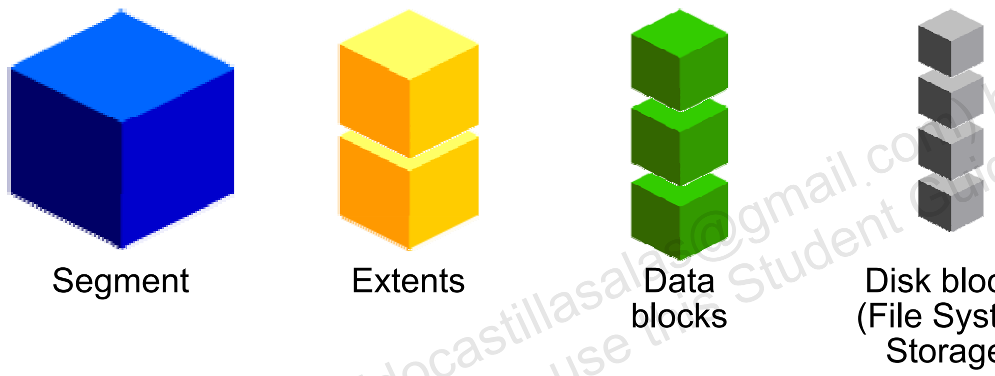
- **Data segments:** Each nonclustered, non-index-organized table has a data segment, with the exception of external tables, global temporary tables, and partitioned tables (in which each table has one or more segments). All of the table's data is stored in the extents of its data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.
- **Index segments:** Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
- **Undo segments:** One `UNDO` tablespace is created for each database instance. This tablespace contains numerous undo segments to temporarily store undo information. The information in an undo segment is used to generate read-consistent database information and, during database recovery, to roll back uncommitted transactions for users.
- **Temporary segments:** Temporary segments are created by the Oracle database when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the database for future use. Specify either a default temporary tablespace for every user, or a default temporary tablespace that is used database-wide.

Note: There are other types of segments not listed here. There are also schema objects such as views, packages, triggers, and so on that are not considered segments even though they are database objects. A segment owns its respective disk space allocation. The other objects exist as rows stored in a system metadata segment.

The Oracle Database server dynamically allocates space. When the existing extents of a segment are full, additional extents are added. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on the disk, and they can come from different data files belonging to the same tablespace.

Segments, Extents, and Blocks

- Segments exist in a tablespace.
- Segments are collections of extents.
- Extents are collections of data blocks.
- Data blocks are mapped to disk blocks.



A subset of database objects such as tables and indexes are stored as segments in tablespaces. Each segment contains one or more extents. An extent consists of contiguous data blocks, which means that each extent can exist only in one data file. Data blocks are the smallest unit of I/O in the database.

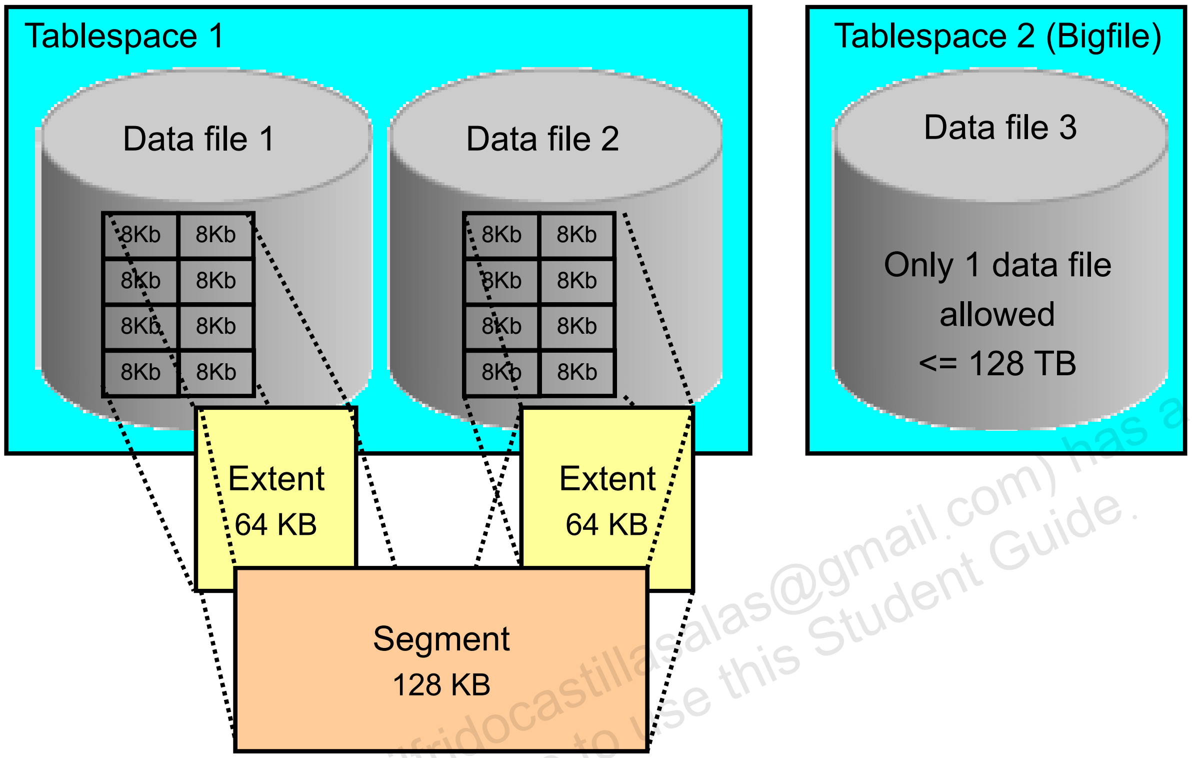
When the database requests a set of data blocks from the operating system (OS), the OS maps this to an actual file system or disk block on the storage device. Because of this, you do not need to know the physical address of any of the data in your database. This also means that a data file can be striped or mirrored on several disks.

The size of the data block can be set at the time of database creation. The default size of 8 KB is adequate for most databases. If your database supports a data warehouse application that has large tables and indexes, a larger block size may be beneficial.

If your database supports a transactional application in which reads and writes are random, specifying a smaller block size may be beneficial. The maximum block size depends on your OS. The minimum Oracle block size is 2 KB; it should rarely (if ever) be used.

You can have tablespaces with a nonstandard block size. For details, see the *Oracle Database Administrator's Guide*.

Tablespaces and Data Files



A database is divided into *tablespaces*, which are logical storage units that can be used to group related logical structures. One or more data files are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace.

The graphic in the slide illustrates tablespace 1, composed of two data files. A segment of 128 KB size, composed of two extents is spanning the two data files. The first extent of size 64 KB is in the first data file and the second extent, also of size 64 KB is in the second data file. Both extents are formed from contiguous 8 KB Oracle blocks.

Note: You can also create bigfile tablespaces, which have only one file that is often very large. The file may be any size up to the maximum that the row ID architecture permits. The maximum size is the block size for the tablespace multiplied by 2^{36}, or 128 TB for a 32 KB block size. Traditional smallfile tablespaces (which are the default) may contain multiple data files, but the files cannot be as large. For more information about bigfile tablespaces, see the *Oracle Database Administrator's Guide*.

SYSTEM and SYSAUX Tablespaces

- The SYSTEM and SYSAUX tablespaces are mandatory tablespaces that are created at the time of database creation. They must be online.
- The SYSTEM tablespace is used for core functionality (for example, data dictionary tables).
- The auxiliary SYSAUX tablespace is used for additional database components.
- The SYSTEM and SYSAUX tablespaces should not be used for application data.

Each Oracle database must contain a SYSTEM tablespace and a SYSAUX tablespace. They are automatically created when the database is created. The system default is to create a smallfile tablespace. You can also create bigfile tablespaces, which enable the Oracle database to manage ultralarge files.

A tablespace can be online (accessible) or offline (not accessible). The SYSTEM tablespace is always online when the database is open. It stores tables that support the core functionality of the database, such as the data dictionary tables.

The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace. The SYSAUX tablespace stores many database components, and it must be online for the correct functioning of all database components. The SYSTEM and SYSAUX tablespaces are not recommended for storing an application's data. Additional tablespaces can be created for this purpose.

Note: The SYSAUX tablespace may be taken offline to perform tablespace recovery, whereas this is not possible for the SYSTEM tablespace. Neither of them may be made read-only.

Oracle Container Database: Introduction

- *Pluggable database*: Is a set of database schemas that appears logically to users and applications as a separate database
- *Multitenant container database*: Has a database instance and database files at the physical level
- All pluggable databases share:
  - Background processes
  - Shared/process memory
  - Oracle metadata

A pluggable database (PDB) is a set of database schemas that appears logically to users and applications as a separate database. But at the physical level, the multitenant container database (CDB) has a database instance and database files, just as a non-container database does.

It is easy to plug non-CDBs into a CDB.

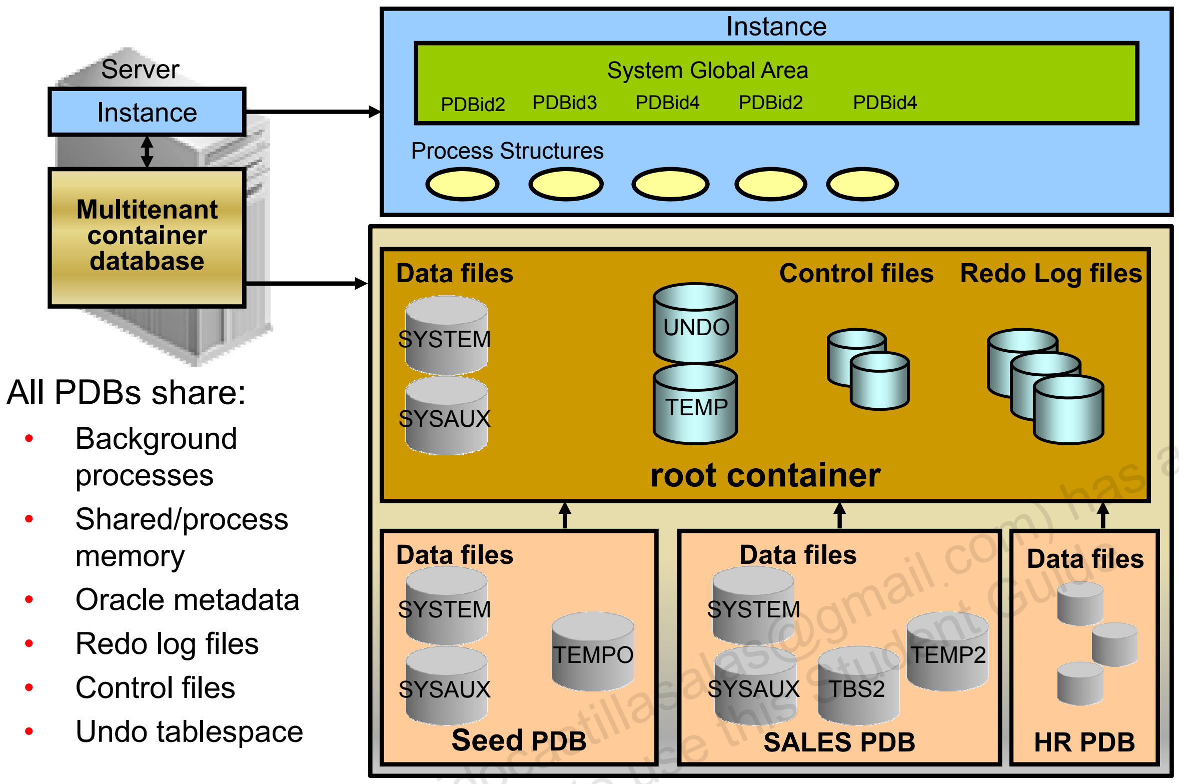
A CDB avoids redundancy of:

- Background processes
- Memory allocation
- Oracle metadata in several data dictionaries

A CDB grouping several applications has one instance, consequently one set of background processes, one SGA allocation and one data dictionary in the root container, common for all PDBs, each PDB maintaining its own application data dictionary.

When applications need to be patched or upgraded, the maintenance operation is performed only once on the CDB and, consequently, all applications are updated at the same time.

Multitenant Architecture



All PDBs share:

- Background processes
- Shared/process memory
- Oracle metadata
- Redo log files
- Control files
- Undo tablespace

The graphic in the slide shows a CDB with four containers: the root, the seed, and two PDBs. The two applications (HR and SALES) use a single instance and are maintained separately.

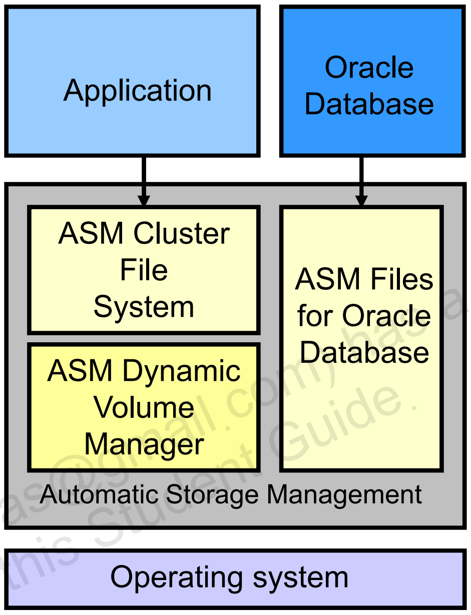
At the physical level, the CDB has a database instance and database files, just as a non-CDB does.

- The redo log files are common for the whole CDB. The information it contains is annotated with the identity of the PDB where a change occurs. Oracle GoldenGate is enhanced to understand the format of the redo log for a CDB. All PDBs in a CDB share the ARCHIVELOG mode of the CDB.
- The control files are common for the whole CDB. The control files are updated to reflect any additional tablespace and data files of plugged PDBs.
- The UNDO tablespace is common for all containers.
- A temporary tablespace common to all containers is required. But each PDB can hold its own temporary tablespace for its own local users.
- Each container has its own data dictionary stored in its proper `SYSTEM` tablespace, containing its own metadata, and a `SYSAUX` tablespace.
- The PDBs can create tablespaces within the PDB according to application needs.
- Each data file is associated with a specific container, named *`CON_ID`*.

Refer to the *Oracle Database 12c: Managing Multitenant* course for detailed information.

Automatic Storage Management

- Is a portable and high-performance cluster file system
- Manages Oracle database files
- Manages application files with ASM Cluster File System (ACFS)
- Spreads data across disks to balance load
- Mirrors data in case of failures
- Solves storage management challenges


Application
Oracle Database
ASM Cluster File System
ASM Files for Oracle Database
ASM Dynamic Volume Manager
Automatic Storage Management
Operating system

Automatic Storage Management (ASM) provides vertical integration of the file system and the volume manager for Oracle database files. ASM can provide management for single symmetric multiprocessing (SMP) machines or across multiple nodes of a cluster for Oracle Real Application Clusters (RAC) support.

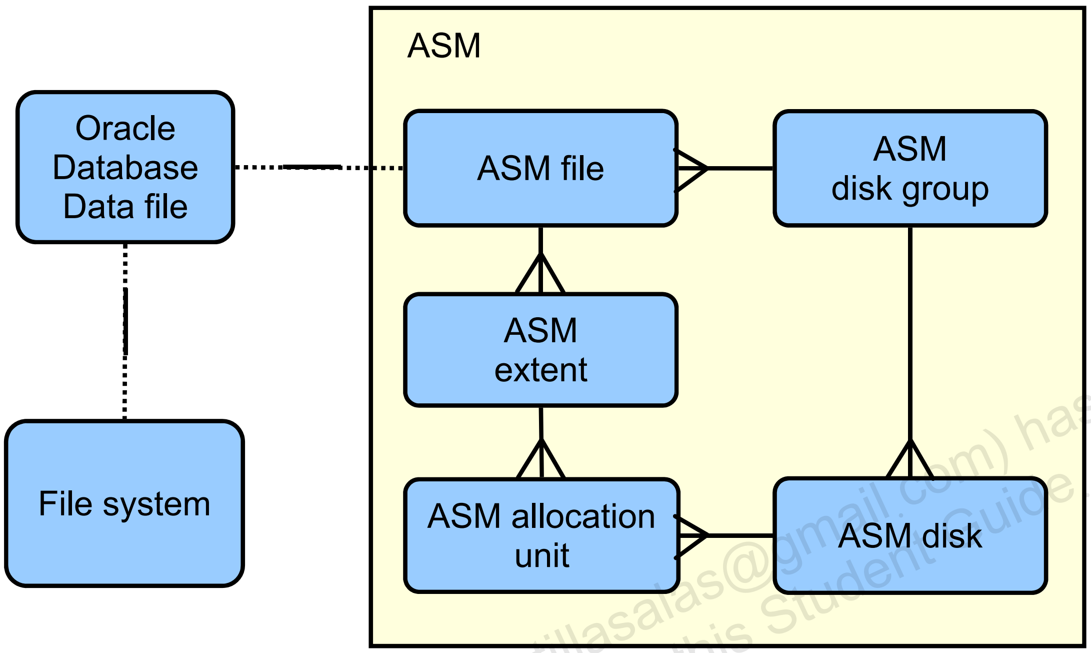
Oracle ASM Cluster File System (ACFS) is a multi-platform, scalable file system, and storage management technology that extends ASM functionality to support application files outside of the Oracle Database such as executables, reports, BFILEs, video, audio, text, images, and other general-purpose application file data.

ASM distributes input/output (I/O) load across all available resources to optimize performance while removing the need for manual I/O tuning. ASM helps database administrators (DBAs) manage a dynamic database environment by enabling them to increase the database size without having to shut down the database to adjust storage allocation.

ASM can maintain redundant copies of data to provide fault tolerance, or it can be built on top of vendor-supplied storage mechanisms. Data management is done by selecting the desired reliability and performance characteristics for classes of data rather than with human interaction on a per-file basis.

ASM capabilities save the DBA’s time by automating manual storage and thereby increasing the administrator’s ability to manage more and larger databases with increased efficiency.

ASM Storage Components

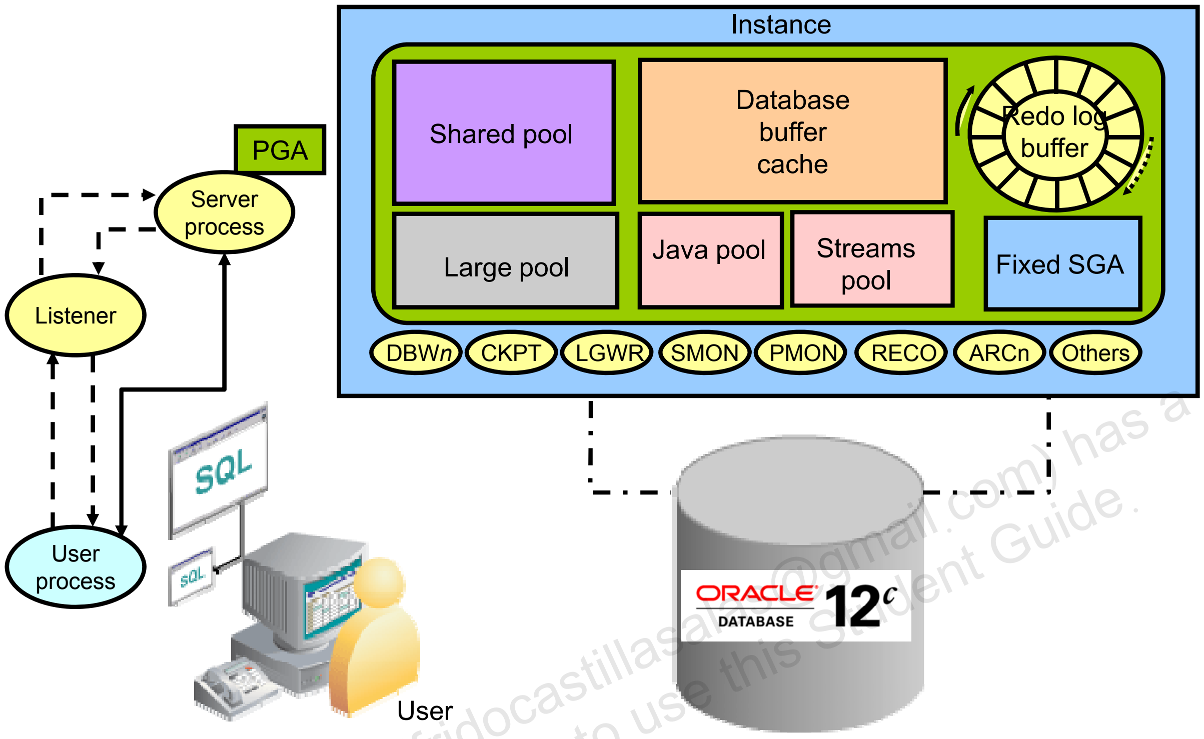


ASM does not eliminate any existing database functionality. Existing databases are able to operate as they always have. New files may be created as ASM files, whereas existing ones are administered in the old way or can be migrated to ASM.

The diagram illustrates the relationships between an Oracle database data file and the ASM storage components. The crow's foot notation represents a one-to-many relationship. An Oracle Database data file has a one-to-one relationship with either a file stored on the operating system in a file system or an ASM file.

An Oracle ASM disk group is a collection of one or more Oracle ASM disks managed as a logical unit. The data structures in a disk group are self-contained using some of the space for metadata needs. Oracle ASM disks are the storage devices provisioned to an Oracle ASM disk group and can be physical disk or partitions, a Logical Unit Number (LUN) from a storage array, a logical volume (LV), or a network-attached file. Each ASM disk is divided into many ASM allocation units, the smallest contiguous amount of disk space that ASM allocates. When you create an ASM disk group, you can set the ASM allocation unit size to 1, 2, 4, 8, 16, 32, or 64 MB depending on the disk group compatibility level. One or more ASM allocation units forms an ASM extent. An Oracle ASM extent is the raw storage used to hold the contents of an Oracle ASM file. An Oracle ASM file consists of one or more file extents. Variable extent sizes of 1*AU size, 4*AU size, and 16*AU size are used for supporting very large ASM files.

Interacting with an Oracle Database: Memory, Processes, and Storage



The following example describes Oracle database operations at the most basic level. It illustrates an Oracle database configuration in which the user and associated server process are on separate computers, connected through a network.

1. An instance has started on a node where Oracle Database is installed, often called the *host* or *database server*.
2. A user starts an application spawning a user process. The application attempts to establish a connection to the server. (The connection may be local, client/server, or a three-tier connection from a middle tier.)
3. The server runs a listener that has the appropriate Oracle Net Services handler. The listener detects the connection request from the application and creates a dedicated server process on behalf of the user process.
4. The user runs a DML-type SQL statement and commits the transaction. For example, the user changes the address of a customer in a table and commits the change.
5. The server process receives the statement and checks the shared pool (an SGA component) for any shared SQL area that contains an identical SQL statement. If a shared SQL area is found, the server process checks the user's access privileges to the requested data, and the existing shared SQL area is used to process the statement. If a shared SQL area is not found, a new shared SQL area is allocated for the statement so that it can be parsed and processed.

6. The server process retrieves any necessary data values, either from the actual data file (table) or from values stored in the database buffer cache.
7. The server process modifies data in the SGA. Because the transaction is committed, the Log Writer process (LGWR) immediately records the transaction in the redo log file. The Database Writer process (DBW*n*) writes modified blocks permanently to disk when it is efficient to do so.
8. If the transaction is successful, the server process sends a message across the network to the application. If it is not successful, an error message is transmitted.
9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

Quiz

The Process Monitor process (PMON):

a. Performs recovery at instance startup
b. Performs process recovery when a user process fails
c. Automatically resolves all in-doubt transactions
d. Writes the redo log buffer to a redo log file

ORACLE

Answer: b

Summary

In this lesson, you should have learned how to:

- List the major architectural components of Oracle Database
- Explain memory structures
- Describe background processes
- Correlate logical and physical storage structures
- Describe pluggable databases
- Describe ASM storage components

Practice: Overview

This is a paper practice with questions about:

- Database architecture
- Memory
- Processes
- File structures