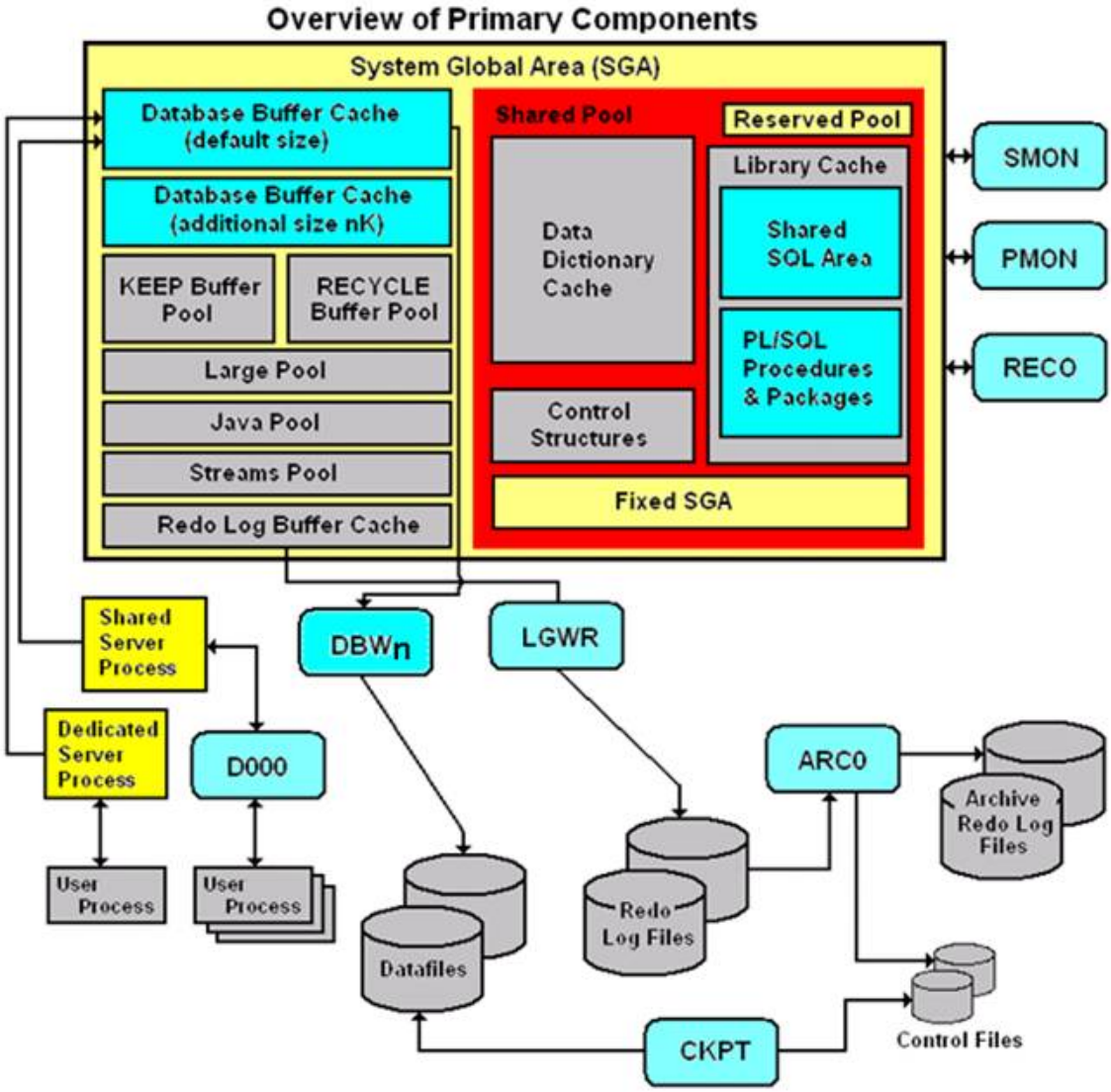
File=module1-architecture.htm; updated 5/13/2013
Figures shown in these notes are from **Oracle® Database Concepts 11*g* Release 2**

Module 1 – Oracle Architecture

Objectives

These notes introduce the Oracle server architecture. The architecture includes physical components, memory components, processes, and logical structures.

Primary Architecture Components



The figure shown above details the Oracle architecture.

Oracle server: An Oracle server includes an **Oracle Instance** and an **Oracle database**.

- An Oracle database includes several different types of files: datafiles, control files, redo log files and archive redo log files. The Oracle server also accesses parameter files and

password files.

- This set of files has several purposes.
  - One is to enable system users to process SQL statements.
  - Another is to improve system performance.
  - Still another is to ensure the database can be recovered if there is a software/hardware failure.
- The database server must manage large amounts of data in a multi-user environment.
- The server must manage concurrent access to the same data.
- The server must deliver high performance. This generally means fast response times.

Oracle instance: An Oracle Instance consists of **two** different sets of components:

- The first component set is the set of **background processes** (PMON, SMON, RECO, DBW0, LGWR, CKPT, D000 and others).
  - These will be covered later in detail – each background process is a computer program.
  - These processes perform input/output and monitor other Oracle processes to provide good performance and database reliability.
- The second component set includes the **memory structures** that comprise the Oracle instance.
  - When an instance starts up, a memory structure called the System Global Area (SGA) is allocated.
  - At this point the background processes also start.
- An Oracle Instance provides access to one and only one Oracle database.

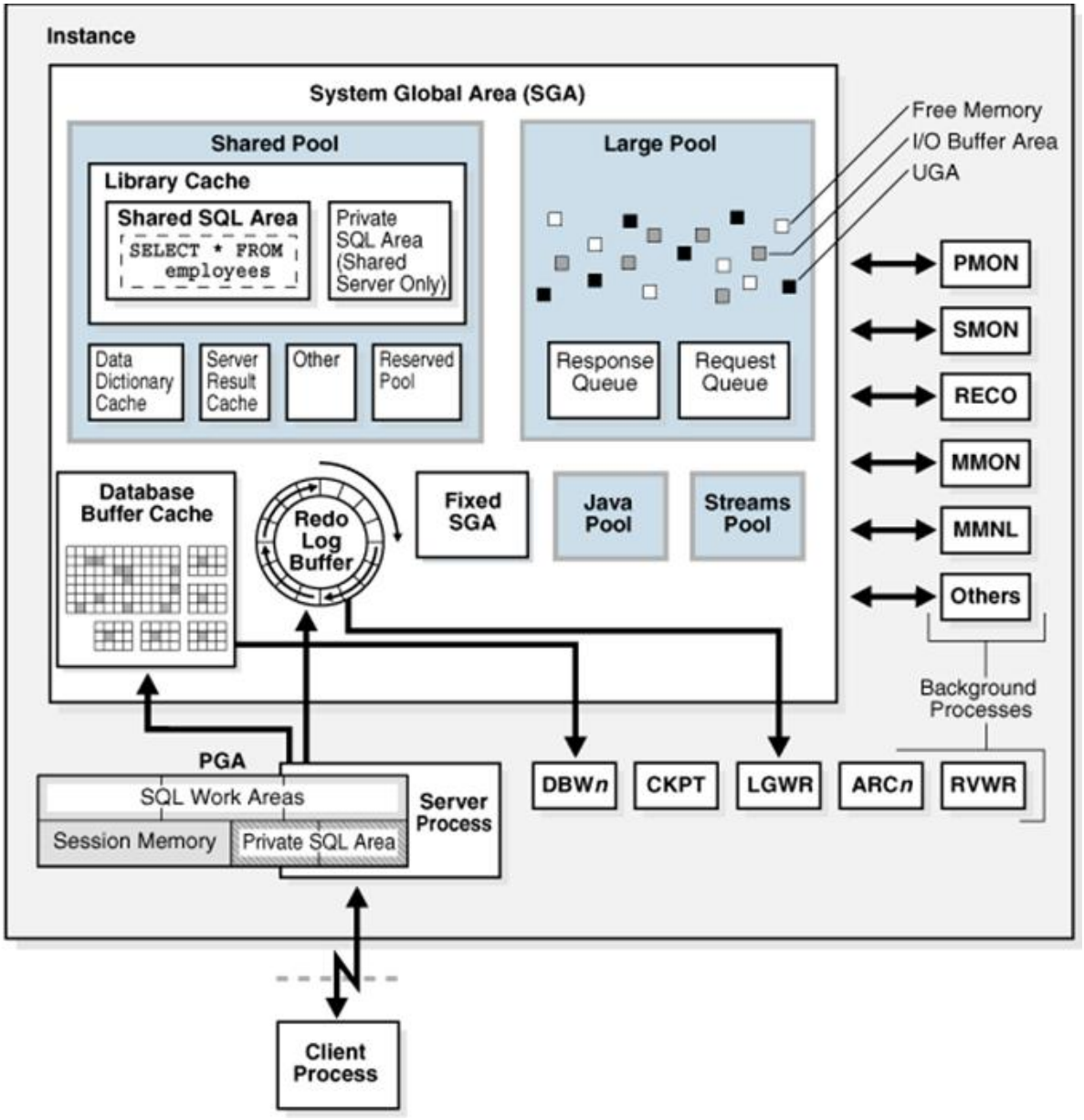
Oracle database: An Oracle database consists of files.

- Sometimes these are referred to as operating system files, but they are actually **database files** that store the database information that a firm or organization needs in order to operate.
- The **redo log files** are used to recover the database in the event of application program failures, instance failures and other minor failures.
- The **archived redo log files** are used to recover the database if a disk fails.
- Other files not shown in the figure include:
  - The required **parameter file** that is used to specify parameters for configuring an Oracle instance when it starts up.
  - The optional **password file** authenticates special users of the database – these are termed **privileged users** and include database administrators.
  - **Alert** and **Trace Log Files** – these files store information about errors and actions taken that affect the configuration of the database.

User and server processes: The processes shown in the figure are called **user** and **server processes**. These processes are used to manage the execution of SQL statements.

- A **Shared Server Process** can share memory and variable processing for multiple user processes.
- A **Dedicated Server Process** manages memory and variables for a single user process.

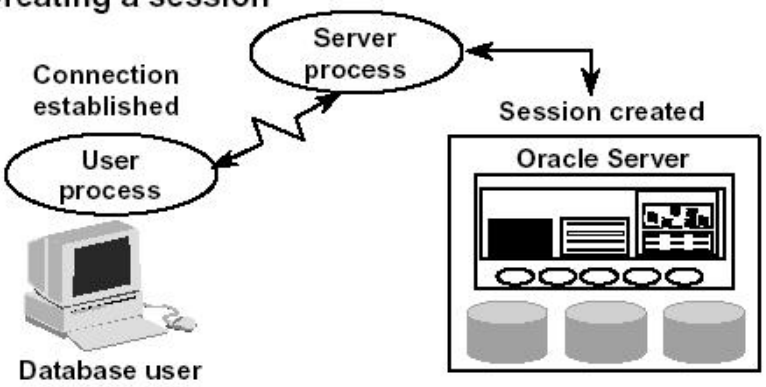
This figure from the *Oracle Database Administration Guide* provides another way of viewing the **SGA**.



Connecting to an Oracle Instance – Creating a Session

Connecting to an Oracle Instance:

- Establishing a user connection
- Creating a session



System users can connect to an Oracle database through SQLPlus or through an application program like the Internet Developer Suite (the program becomes the system user). This connection enables users to execute SQL statements.

The act of connecting creates a communication pathway between a user process and an Oracle Server. As is shown in the figure above, the User Process communicates with the Oracle Server through a Server Process. The User Process executes on the client computer. The Server Process executes on the server computer, and actually executes SQL statements submitted by the system user.

The figure shows a one-to-one correspondence between the User and Server Processes. This is called a **Dedicated Server** connection. An alternative configuration is to use a **Shared Server** where more than one User Process shares a Server Process.

Sessions: When a user connects to an Oracle server, this is termed a session. The **User Global Area** is session memory and these memory structures are described later in this document. The session starts when the Oracle server validates the user for connection. The session ends when the user logs out (disconnects) or if the connection terminates abnormally (network failure or client computer failure).

A user can typically have more than one concurrent session, e.g., the user may connect using SQLPlus and also connect using Internet Developer Suite tools at the same time. The limit of concurrent session connections is controlled by the DBA.

If a system users attempts to connect and the Oracle Server is not running, the system user receives the **Oracle Not Available** error message.

Physical Structure – Database Files

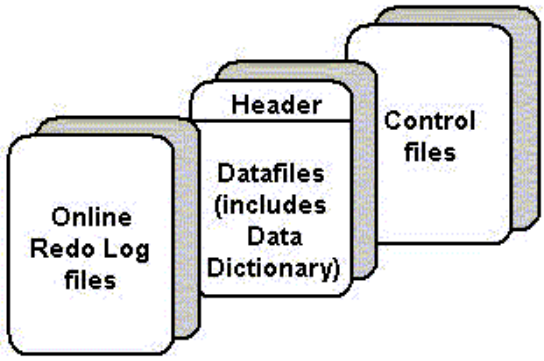
As was noted above, an Oracle database consists of physical files. The database itself has:

- **Datafiles** – these contain the organization's actual data.
- **Redo log files** – these contain a chronological record of changes made to the database, and enable recovery when failures occur.
- **Control files** – these are used to synchronize all database activities and are covered in more detail in a later module.

Physical Structure

The physical structure includes three types of files:

- **Control files**
- **Datafiles**
- **Redo log files**



Other key files as noted above include:

- Parameter file – there are two types of parameter files.
  - The **init.ora** file (also called the **PFILE**) is a **static parameter file**. It contains parameters that specify how the database instance is to start up. For example, some parameters will specify how to allocate memory to the various parts of the system global area.
  - The **spfile.ora** is a **dynamic parameter file**. It also stores parameters to specify how to startup a database; however, its parameters can be modified while the database is running.
- Password file – specifies which *special* users are authenticated to startup/shut down an Oracle Instance.
- Archived redo log files – these are copies of the redo log files and are necessary for recovery in an online, transaction-processing environment in the event of a disk failure.

Memory Management and Memory Structures

Oracle Database Memory Management

Memory management - focus is to maintain optimal sizes for memory structures.

- Memory is managed based on memory-related **initialization parameters**.
- These values are stored in the init.ora file for each database.

Three basic options for memory management are as follows:

- **Automatic memory management**:
  - DBA specifies the target size for instance memory.
  - The database instance automatically tunes to the target memory size.
  - Database redistributes memory as needed between the SGA and the instance PGA.

- **Automatic shared memory management**:
  - This management mode is partially automated.
  - DBA specifies the target size for the SGA.

- DBA can optionally set an aggregate target size for the PGA or managing PGA work areas individually.

- **Manual memory management**:
  - Instead of setting the total memory size, the DBA sets 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.

The memory structures include three areas of memory:
- System Global Area (SGA) – this is allocated when an Oracle Instance starts up.
- Program Global Area (PGA) – this is allocated when a Server Process starts up.
- User Global Area (UGA) – this is allocated when a user connects to create a session.

System Global Area

The **SGA** is a read/write memory area that stores information shared by all database processes and by all users of the database (sometimes it is called the **Shared Global Area**).
- This information includes both organizational data and control information used by the Oracle Server.
- The SGA is allocated in memory and virtual memory.
- The size of the SGA can be established by a DBA by assigning a value to the parameter **SGA_MAX_SIZE** in the parameter file—this is an optional parameter.

The SGA is allocated when an Oracle instance (database) is started up based on values specified in the initialization parameter file (either PFILE or SPFILE).

The SGA has the following mandatory memory structures:
- Database Buffer Cache
- Redo Log Buffer
- Java Pool
- Streams Pool
- Shared Pool – includes two components:
  - Library Cache
  - Data Dictionary Cache
- Other structures (for example, lock and latch management, statistical data)

Additional optional memory structures in the SGA include:
- Large Pool

The **SHOW SGA** SQL command will show you the SGA memory allocations.
- This is a recent clip of the SGA for the DBORCL database at SIUE.
- In order to execute SHOW SGA you must be connected with the special privilege **SYSDBA** (which is only available to user accounts that are members of the DBA Linux group).

```
SQL> connect / as sysdba
Connected.
SQL> show sga
```

```
Total System Global Area 1610612736 bytes
Fixed Size                  2084296 bytes
Variable Size            1006633528 bytes
Database Buffers          587202560 bytes
Redo Buffers               14692352 bytes
```

Early versions of Oracle used a **Static SGA**. This meant that if modifications to memory management were required, the database had to be shutdown, modifications were made to the **init.ora** parameter file, and then the database had to be restarted.

Oracle 11g uses a **Dynamic SGA**. Memory configurations for the system global area can be made without shutting down the database instance. The DBA can resize the Database Buffer Cache and Shared Pool dynamically.

Several initialization parameters are set that affect the amount of random access memory dedicated to the SGA of an Oracle Instance. These are:

- **SGA_MAX_SIZE**: This optional parameter is used to set a limit on the amount of **virtual memory** allocated to the SGA – a typical setting might be **1 GB**; however, if the value for SGA_MAX_SIZE in the initialization parameter file or server parameter file is less than the sum the memory allocated for all components, either explicitly in the parameter file or by default, at the time the instance is initialized, then the database ignores the setting for SGA_MAX_SIZE. For optimal performance, the entire SGA should fit in real memory to eliminate paging to/from disk by the operating system.
- **DB_CACHE_SIZE**: This optional parameter is used to tune the amount memory allocated to the Database Buffer Cache in standard database blocks. Block sizes vary among operating systems. The DBORCL database uses **8 KB** blocks. The total blocks in the cache defaults to **48 MB** on LINUX/UNIX and **52 MB** on Windows operating systems.
- **LOG_BUFFER**: This optional parameter specifies the number of bytes allocated for the Redo Log Buffer.
- **SHARED_POOL_SIZE**: This optional parameter specifies the number of bytes of memory allocated to shared SQL and PL/SQL. The default is **16 MB**. If the operating system is based on a **64 bit** configuration, then the default size is **64 MB**.
- **LARGE_POOL_SIZE**: This is an optional memory object – the size of the Large Pool defaults to zero. If the init.ora parameter **PARALLEL_AUTOMATIC_TUNING** is set to **TRUE**, then the default size is automatically calculated.
- **JAVA_POOL_SIZE**: This is another optional memory object. The default is **24 MB** of memory.

The size of the SGA cannot exceed the parameter **SGA_MAX_SIZE** minus the combination of the size of the additional parameters, **DB_CACHE_SIZE**, **LOG_BUFFER**, **SHARED_POOL_SIZE**, **LARGE_POOL_SIZE**, and **JAVA_POOL_SIZE**.

Memory is allocated to the SGA as contiguous virtual memory in units termed granules. Granule size depends on the estimated total size of the SGA, which as was noted above, depends on the SGA_MAX_SIZE parameter. Granules are sized as follows:

- If the SGA is less than **1 GB** in total, each granule is **4 MB**.
- If the SGA is greater than **1 GB** in total, each granule is **16 MB**.

Granules are assigned to the Database Buffer Cache, Shared Pool, Java Pool, and other memory structures, and these memory components can dynamically grow and shrink. Using contiguous memory improves system performance. The actual number of granules assigned to one of these memory components can be determined by querying the database view named **V$BUFFER_POOL**.

Granules are allocated when the Oracle server starts a database instance in order to provide memory addressing space to meet the SGA_MAX_SIZE parameter. The minimum is 3 granules: one each for the fixed SGA, Database Buffer Cache, and Shared Pool. In practice, you'll find the SGA is allocated much more memory than this. The SELECT statement shown below shows a current_size of 1,152 granules.

```
SELECT name, block_size, current_size, prev_size, prev_buffers
FROM v$buffer_pool;

NAME                 BLOCK_SIZE CURRENT_SIZE  PREV_SIZE PREV_BUFFERS
-------------------- ---------- ------------ ---------- ------------
DEFAULT                    8192          560        576        71244
```

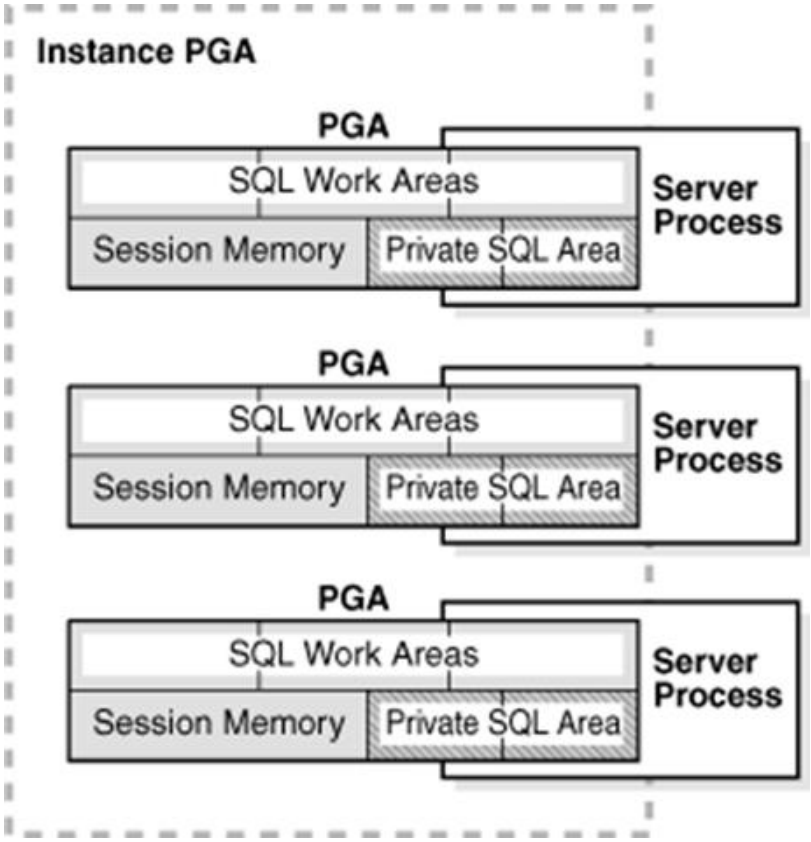
For additional information on the dynamic SGA sizing, enroll in Oracle's *Oracle11g Database Performance Tuning* course.

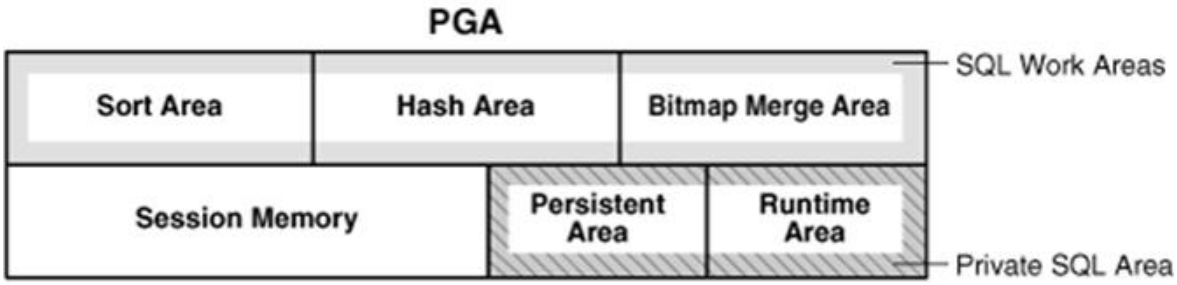
Program Global Area (PGA)

A PGA is:

- a **nonshared** memory region that contains data and control information exclusively for use by an Oracle process.
- A PGA is created by Oracle Database when an Oracle process is started.
- One PGA exists for each **Server Process** and each **Background Process**. It stores data and control information for a single **Server Process** or a single **Background Process**.
- It is allocated when a process is created and the memory is scavenged by the operating system when the process terminates. This is **NOT** a shared part of memory – one PGA to each process only.
- 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.

The **Program Global Area** is also termed the **Process Global Area (PGA**) and is a part of memory allocated that is outside of the **Oracle Instance**.





The content of the PGA varies, but as shown in the figure above, generally includes the following:

- **Private SQL Area**: Stores information for a parsed SQL statement – stores bind variable values and runtime memory allocations. A user session issuing SQL statements has a Private SQL Area that may be associated with a Shared SQL Area if the same SQL statement is being executed by more than one system user. This often happens in OLTP environments where many users are executing and using the same application program.
  - **Dedicated Server environment** – the Private SQL Area is located in the Program Global Area.
  - **Shared Server environment** – the Private SQL Area is located in the System Global Area.

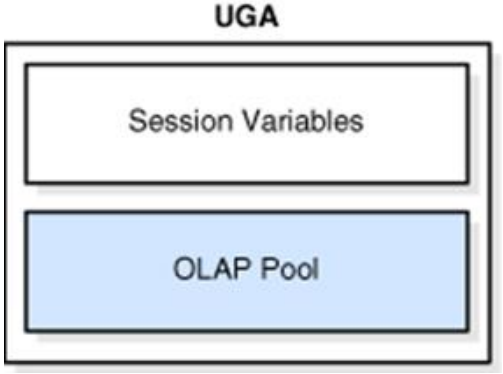
- **Session Memory**: Memory that holds session variables and other session information.

- **SQL Work Areas**: Memory allocated for sort, hash-join, bitmap merge, and bitmap create types of operations.
  - Oracle 9i and later versions enable automatic sizing of the SQL Work Areas by setting the **WORKAREA_SIZE_POLICY = AUTO** parameter (this is the default!) and **PGA_AGGREGATE_TARGET = n** (where n is some amount of memory established by the

DBA). However, the DBA can let the Oracle DBMS determine the appropriate amount of memory.

User Global Area

The **User Global Area** is session memory.



A session that loads a PL/SQL **package** into memory has the **package state** stored to the UGA. The **package state** is the set of values stored in all the package variables at a specific time. The state changes as program code the variables. By default, 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.

> ***Note:*** **Oracle OLAP** is a multidimensional analytic engine embedded in Oracle Database 11*g*. Oracle OLAP cubes deliver sophisticated calculations using simple SQL queries - producing results with speed of thought response times.

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.

Automatic Shared Memory Management

Prior to Oracle 10G, a DBA had to manually specify SGA Component sizes through the initialization parameters, such as SHARED_POOL_SIZE, DB_CACHE_SIZE, JAVA_POOL_SIZE, and LARGE_POOL_SIZE parameters.

Automatic Shared Memory Management enables a DBA to specify the total SGA memory available through the **SGA_TARGET** initialization parameter. The Oracle Database automatically distributes this memory among various subcomponents to ensure most effective memory utilization.

The **DBORCL** database **SGA_TARGET** is set in the **initDBORCL.ora** file:

sga_target=1610612736

With automatic SGA memory management, the different SGA components are flexibly sized to adapt to the SGA available.

Setting a single parameter simplifies the administration task – the DBA only specifies the amount of SGA memory available to an instance – the DBA can forget about the sizes of individual components. No out of memory errors are generated unless the system has actually run out of memory. No manual tuning effort is needed.

The **SGA_TARGET** initialization parameter reflects the total size of the SGA and includes memory for the following components:

- Fixed SGA and other internal allocations needed by the Oracle Database instance
- The log buffer
- The shared pool
- The Java pool
- The buffer cache
- The keep and recycle buffer caches (if specified)
- Nonstandard block size buffer caches (if specified)
- The Streams Pool

If **SGA_TARGET** is set to a value greater than **SGA_MAX_SIZE** at startup, then the SGA_MAX_SIZE value is bumped up to accommodate SGA_TARGET.
When you set a value for **SGA_TARGET**, Oracle Database 11*g* automatically sizes the most commonly configured components, including:

- The shared pool (for SQL and PL/SQL execution)
- The Java pool (for Java execution state)
- The large pool (for large allocations such as RMAN backup buffers)
- The buffer cache

There are a few SGA components whose sizes are not automatically adjusted. The DBA must specify the sizes of these components explicitly, if they are needed by an application. Such components are:

- Keep/Recycle buffer caches (controlled by **DB_KEEP_CACHE_SIZE** and **DB_RECYCLE_CACHE_SIZE**)
- Additional buffer caches for non-standard block sizes (controlled by **DB_*n*K_CACHE_SIZE**, *n* = {2, 4, 8, 16, 32})
- Streams Pool (controlled by the new parameter **STREAMS_POOL_SIZE**)

The granule size that is currently being used for the SGA for each component can be viewed in the view **V$SGAINFO**. The size of each component and the time and type of the last resize operation performed on each component can be viewed in the view **V$SGA_DYNAMIC_COMPONENTS**.

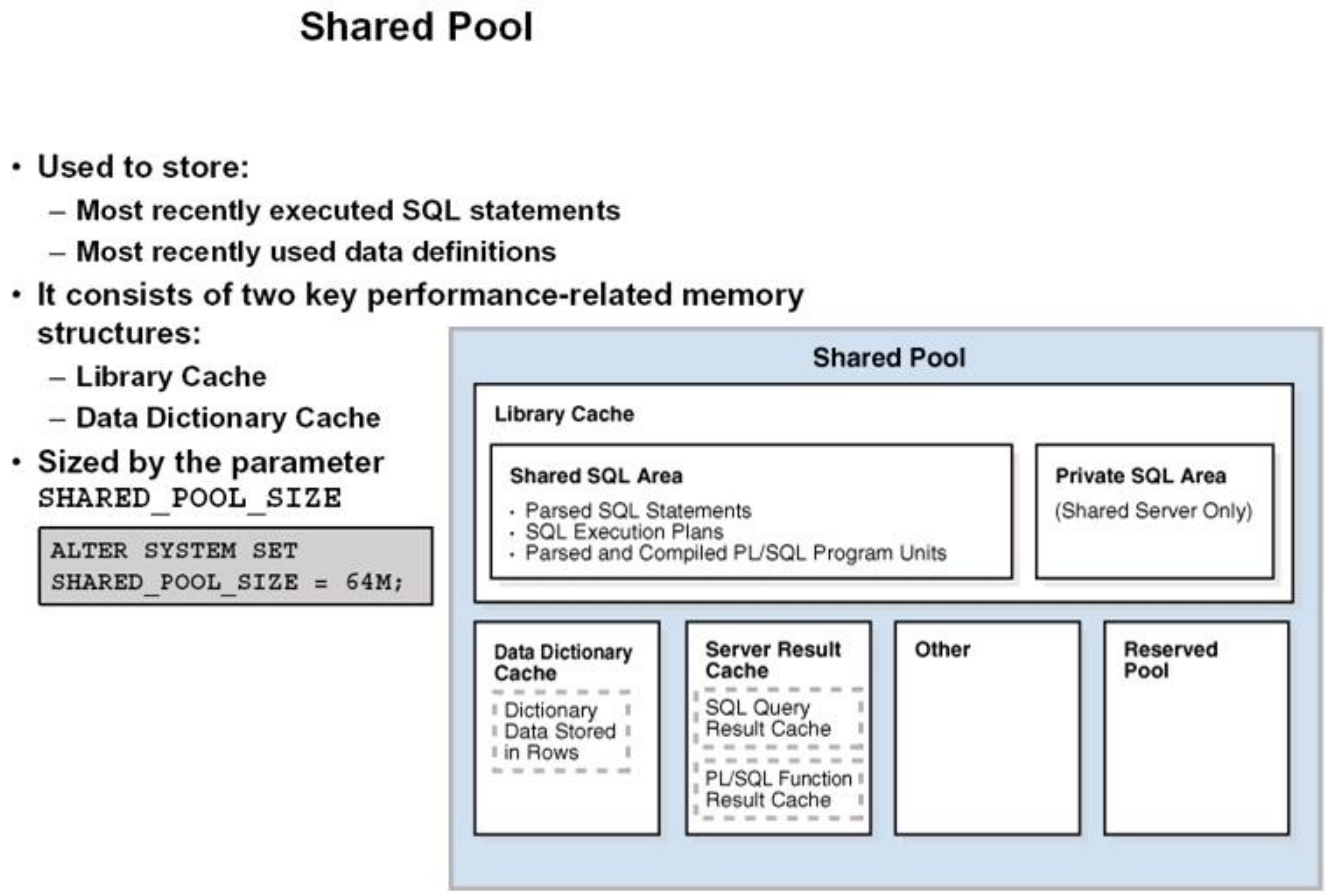
```
SQL> select * from v$sgainfo;
More...

NAME                                  BYTES RES
------------------------------ ---------- ---
Fixed SGA Size                      2084296 No
```

```
Redo Buffers                          14692352 No
Buffer Cache Size                    587202560 Yes
Shared Pool Size                     956301312 Yes
Large Pool Size                       16777216 Yes
Java Pool Size                        33554432 Yes93
Streams Pool Size                            0 Yes
Granule Size                          16777216 No
Maximum SGA Size                    1610612736 No
Startup overhead in Shared Pool       67108864 No
Free SGA Memory Available                    0

11 rows selected.
```

Shared Pool



The **Shared Pool** is a memory structure that is shared by all system users.

- It 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.
- It consists of both fixed and variable structures.
- The variable component grows and shrinks depending on the demands placed on memory

size by system users and application programs.

Memory can be allocated to the Shared Pool by the parameter **SHARED_POOL_SIZE** in the parameter file. The default value of this parameter is **8MB** on 32-bit platforms and **64MB** on 64-bit platforms. Increasing the value of this parameter increases the amount of memory reserved for the shared pool.

You can alter the size of the shared pool dynamically with the **ALTER SYSTEM SET** command. An example command is shown in the figure below. You must keep in mind that the total memory allocated to the SGA is set by the **SGA_TARGET** parameter (and may also be limited by the **SGA_MAX_SIZE** if it is set), and since the Shared Pool is part of the SGA, you cannot exceed the maximum size of the SGA. It is recommended to let Oracle optimize the Shared Pool size.

The Shared Pool stores the most recently executed SQL statements and used data definitions. This is because some system users and application programs will tend to execute the same SQL statements often. Saving this information in memory can improve system performance.

The Shared Pool includes several cache areas described below.

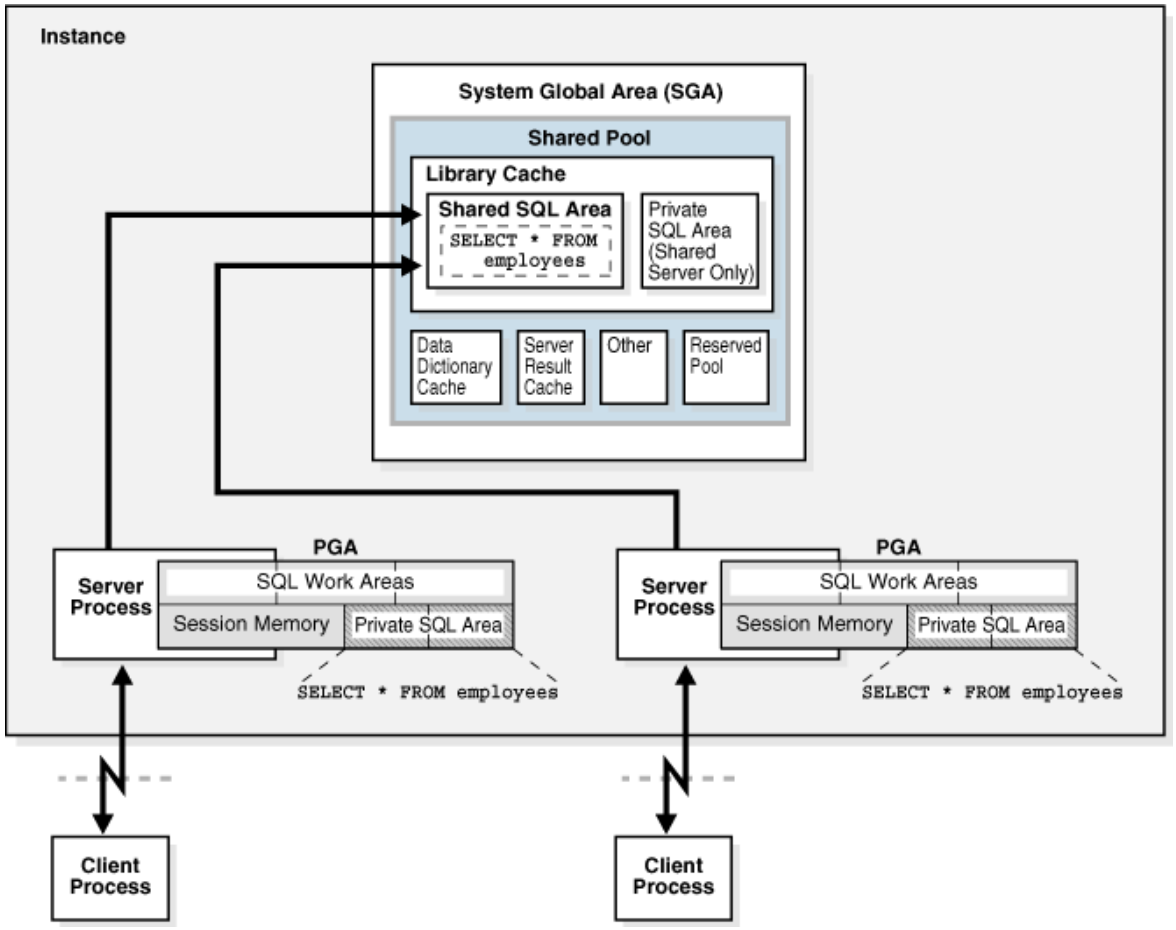
Library Cache

Memory is allocated to the **Library Cache** whenever an SQL statement is parsed or a program unit is called. This enables storage of the most recently used SQL and PL/SQL statements.

If the Library Cache is too small, the Library Cache must purge statement definitions in order to have space to load new SQL and PL/SQL statements. Actual management of this memory structure is through a **Least-Recently-Used (LRU) algorithm**. This means that the SQL and PL/SQL statements that are oldest and least recently used are purged when more storage space is needed.

The Library Cache is composed of two memory subcomponents:

- **Shared SQL**: This stores/shares the execution plan and parse tree for SQL statements, as well as PL/SQL statements such as functions, packages, and triggers. If a system user executes an identical statement, then the statement does not have to be parsed again in order to execute the statement.
- **Private SQL Area**: With a shared server, each session issuing a SQL statement has a private SQL area in its PGA.
    - Each user that submits the same statement has a private SQL area pointing to the same shared SQL area.
    - Many private SQL areas in separate PGAs can be associated with the same shared SQL area.
    - This figure depicts two different client processes issuing the same SQL statement – the parsed solution is already in the Shared SQL Area.



Data Dictionary Cache

The Data Dictionary Cache is a memory structure that caches data dictionary information that has been recently used.

- This cache is necessary because the data dictionary is accessed so often.
- Information accessed includes user account information, datafile names, table descriptions, user privileges, and other information.

The database server manages the size of the Data Dictionary Cache internally and the size depends on the size of the Shared Pool in which the Data Dictionary Cache resides. If the size is too small, then the data dictionary tables that reside on disk must be queried often for information and this will slow down performance.

Server Result Cache

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.

SQL Query Result Cache

This cache stores the results of queries and query fragments.

- Using the cache results for future queries tends to improve performance.
- 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.

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 specify that results be cached.

Buffer Caches

A number of buffer caches are maintained in memory in order to improve system response time.

Database Buffer Cache

The **Database Buffer Cache** is a fairly large memory object that stores the actual data blocks that are retrieved from datafiles by system queries and other data manipulation language commands.

The purpose is to optimize physical input/output of data.

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.
- Database Smart Flash Cache is available only in Solaris and Oracle Enterprise Linux.

A query causes a **Server Process** to look for data.

- The first look is in the Database Buffer Cache to determine if the requested information happens to already be located in memory – thus the information would not need to be retrieved from disk and this would speed up performance.
- If the information is not in the Database Buffer Cache, the Server Process retrieves the information from disk and stores it to the cache.
- Keep in mind that information read from disk is read a **block at a time**, **NOT** a **row at a time**, because a database block is the smallest addressable storage space on disk.

Database blocks are kept in the Database Buffer Cache according to a **Least Recently Used (LRU) algorithm** and are aged out of memory if a buffer cache block is not used in order to provide space for the insertion of newly needed database blocks.

There are three buffer states:

- **Unused** - a buffer is available for use - it has never been used or is currently unused.
- **Clean** - a buffer that was used earlier - the data has been written to disk.
- **Dirty** - a buffer that has modified data that has not been written to disk.

Each buffer has one of two access modes:

- **Pinned** - a buffer is pinned so it does not age out of memory.
- **Free** (unpinned).

The buffers in the cache are organized in two lists:

- the write list and,
- the least recently used (LRU) list.

The **write list** (also called a **write queue**) holds dirty buffers – these are buffers that hold that data that has been modified, but the blocks have not been written back to disk.

The **LRU list** holds unused, free clean buffers, pinned buffers, and free dirty buffers that have not yet been moved to the write list. **Free clean buffers** do not contain any useful data and are available for use. **Pinned buffers** are currently being accessed.

When an Oracle process accesses a buffer, the process moves the buffer to the **most recently used (MRU)** end of the LRU list – this causes dirty buffers to age toward the LRU end of the LRU list.

When an Oracle user process needs a data row, it searches for the data in the database buffer cache because memory can be searched more quickly than hard disk can be accessed. If the data row is already in the cache (a **cache hit**), the process reads the data from memory; otherwise a **cache miss** occurs and data must be read from hard disk into the database buffer cache.

Before reading a data block into the cache, the process must first find a free buffer. The process searches the LRU list, starting at the LRU end of the list. The search continues until a free buffer is found or until the search reaches the threshold limit of buffers.

Each time a user process finds a dirty buffer as it searches the LRU, that buffer is moved to the write list and the search for a free buffer continues.

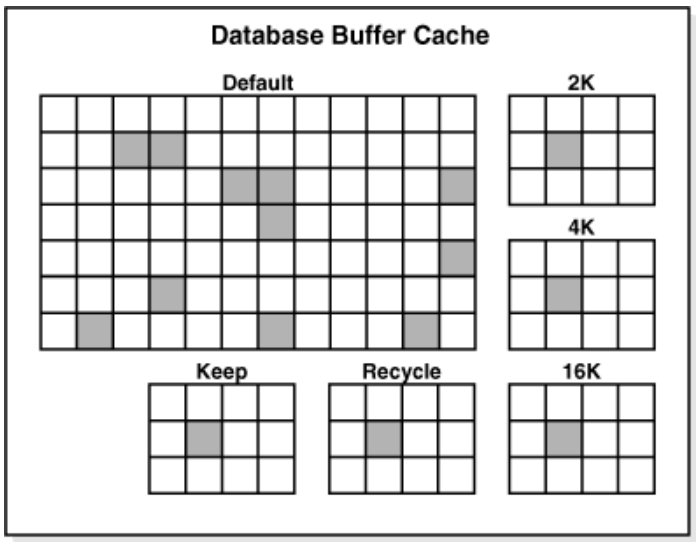
When a user process finds a free buffer, it reads the data block from disk into the buffer and moves the buffer to the MRU end of the LRU list.

If an Oracle user process searches the threshold limit of buffers without finding a free buffer, the process stops searching the LRU list and signals the DBWn background process to write some of the dirty buffers to disk. This frees up some buffers.

Database Buffer Cache Block Size

The block size for a database is set when a database is created and is determined by the init.ora parameter file parameter named **DB_BLOCK_SIZE**.

- Typical block sizes are **2KB**, **4KB**, **8KB**, **16KB**, and **32KB**.
- The size of blocks in the Database Buffer Cache matches the block size for the database.
- The **DBORCL** database uses an **8KB** block size.
- This figure shows that the use of non-standard block sizes results in multiple database buffer cache memory allocations.



Because tablespaces that store oracle tables can use different (non-standard) block sizes, there can be more than one Database Buffer Cache allocated to match block sizes in the cache with the block sizes in the non-standard tablespaces.

The size of the Database Buffer Caches can be controlled by the parameters **DB_CACHE_SIZE** and **DB_nK_CACHE_SIZE** to dynamically change the memory allocated to the caches without restarting the Oracle instance.

You can dynamically change the size of the Database Buffer Cache with the ALTER SYSTEM command like the one shown here:

```
ALTER SYSTEM SET DB_CACHE_SIZE = 96M;
```

You can have the Oracle Server gather statistics about the Database Buffer Cache to help you size it to achieve an optimal workload for the memory allocation. This information is displayed from the **V$DB_CACHE_ADVICE** view. In order for statistics to be gathered, you can dynamically alter the system by using the **ALTER SYSTEM SET DB_CACHE_ADVICE (OFF, ON, READY)** command. However, gathering statistics on system performance always incurs some overhead that will slow down system performance.

```
SQL> ALTER SYSTEM SET db_cache_advice = ON;

System altered.

SQL> DESC V$DB_cache_advice;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 ID                                                 NUMBER
 NAME                                               VARCHAR2(20)
 BLOCK_SIZE                                         NUMBER
 ADVICE_STATUS                                      VARCHAR2(3)
 SIZE_FOR_ESTIMATE                                  NUMBER
 SIZE_FACTOR                                        NUMBER
 BUFFERS_FOR_ESTIMATE                               NUMBER
 ESTD_PHYSICAL_READ_FACTOR                          NUMBER
```

```
 ESTD_PHYSICAL_READS                                    NUMBER
 ESTD_PHYSICAL_READ_TIME                                NUMBER
 ESTD_PCT_OF_DB_TIME_FOR_READS                          NUMBER
 ESTD_CLUSTER_READS                                     NUMBER
 ESTD_CLUSTER_READ_TIME                                 NUMBER

SQL> SELECT name, block_size, advice_status FROM v$db_cache_advice;

NAME                 BLOCK_SIZE ADV
-------------------- ---------- ---
DEFAULT                    8192 ON
<more rows will display>
21 rows selected.

SQL> ALTER SYSTEM SET db_cache_advice = OFF;

System altered.
```

KEEP Buffer Pool

This pool retains blocks in memory (data from tables) that are likely to be reused throughout daily processing. An example might be a table containing user names and passwords or a validation table of some type.

The **DB_KEEP_CACHE_SIZE** parameter sizes the KEEP Buffer Pool.

RECYCLE Buffer Pool

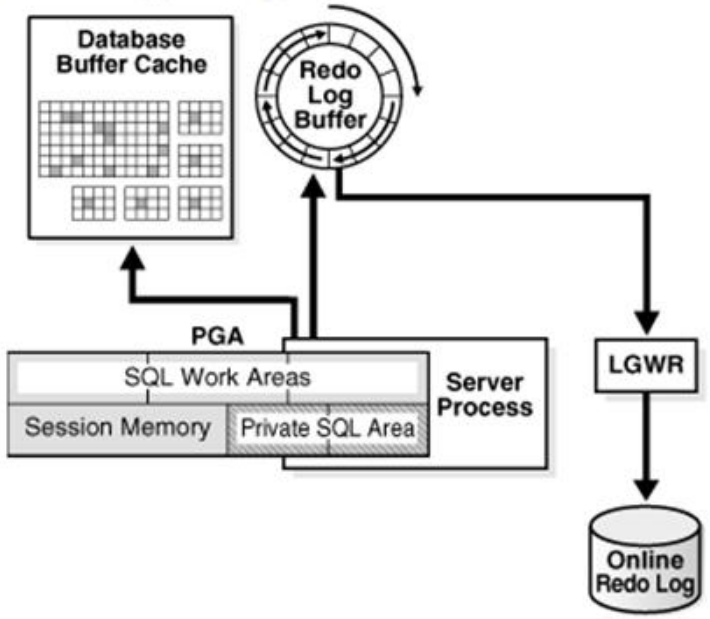
This pool is used to store table data that is unlikely to be reused throughout daily processing – thus the data blocks are quickly removed from memory when not needed.

The **DB_RECYCLE_CACHE_SIZE** parameter sizes the Recycle Buffer Pool.

Redo Log Buffer

Redo Log Buffer

- **Records all changes made to the database data blocks**
- **Primary purpose is recovery**
- **Changes recorded within are called redo entries**
- **Redo entries contain information to reconstruct or redo changes**
- **Size defined by `LOG_BUFFER`**



The **Redo Log Buffer** memory object stores images of all changes made to database blocks.

- Database blocks typically store several table rows of organizational data. This means that if a single column value from one row in a block is changed, the block image is stored. Changes include INSERT, UPDATE, DELETE, CREATE, ALTER, or DROP.
- 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 Redo Log Buffer as a circular buffer that is reused over and over. As the buffer fills up, copies of the images are stored to the **Redo Log Files** that are covered in more detail in a later module.

Large Pool

The **Large Pool** is an optional memory structure that primarily relieves the memory burden placed on the Shared Pool. The Large Pool is used for the following tasks if it is allocated:

- Allocating space for session memory requirements from the User Global Area where a Shared Server is in use.

- Transactions that interact with more than one database, e.g., a distributed database scenario.
- Backup and restore operations by the Recovery Manager (RMAN) process.
  - RMAN uses this only if the **BACKUP_DISK_IO = n** and **BACKUP_TAPE_IO_SLAVE = TRUE** parameters are set.
  - If the Large Pool is too small, memory allocation for backup will fail and memory will be allocated from the Shared Pool.
- Parallel execution message buffers for parallel server operations. The **PARALLEL_AUTOMATIC_TUNING = TRUE** parameter must be set.

The Large Pool size is set with the **LARGE_POOL_SIZE** parameter – this is not a dynamic parameter. It does not use an LRU list to manage memory.

Java Pool

The Java Pool is an **optional** memory object, but is required if the database has Oracle Java installed and in use for Oracle JVM (Java Virtual Machine).

- The size is set with the **JAVA_POOL_SIZE** parameter that defaults to 24MB.
- The Java Pool is used for memory allocation to parse Java commands and to store data associated with Java commands.
- Storing Java code and data in the Java Pool is analogous to SQL and PL/SQL code cached in the Shared Pool.

Streams Pool

This pool stores data and control structures to support the Oracle Streams feature of Oracle Enterprise Edition.

- Oracle Steams manages sharing of data and events in a distributed environment.
- It is sized with the parameter **STREAMS_POOL_SIZE**.
- If STEAMS_POOL_SIZE is not set or is zero, the size of the pool grows dynamically.

Processes

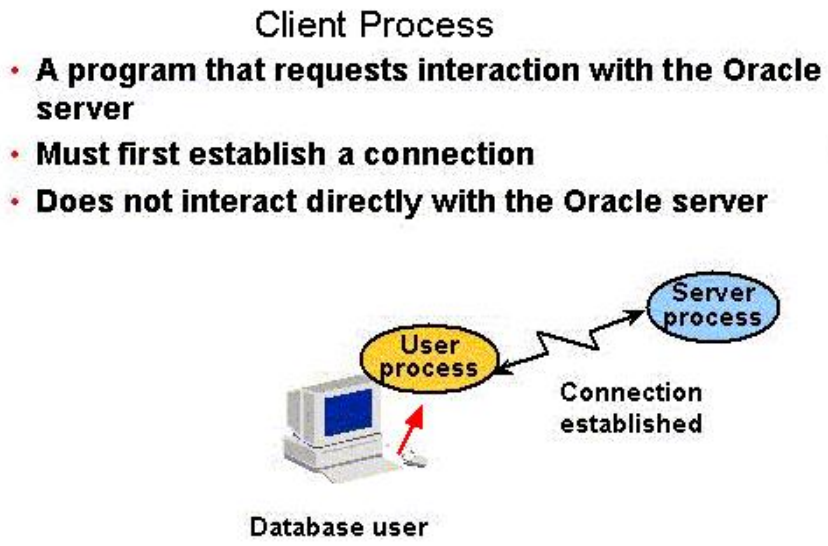
You need to understand three different types of Processes:

- **User Process**: Starts when a database user requests to connect to an Oracle Server.
- **Server Process**: Establishes the Connection to an Oracle Instance when a User Process requests connection – makes the connection for the User Process.
- **Background Processes**: These start when an Oracle Instance is started up.

Client Process

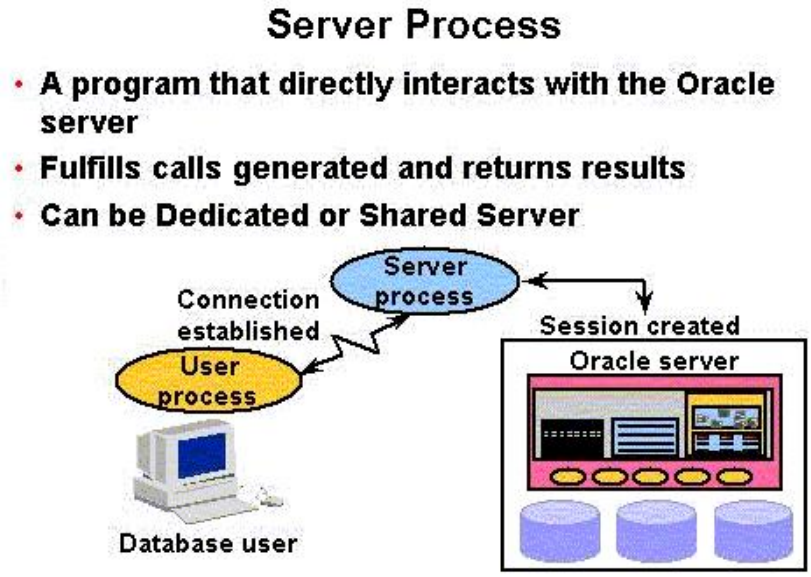
In order to use Oracle, you must connect to the database. This must occur whether you're using SQLPlus, an Oracle tool such as Designer or Forms, or an application program. The client process

is also termed the user process in some Oracle documentation.



This generates a User Process (a memory object) that generates programmatic calls through your user interface (SQLPlus, Integrated Developer Suite, or application program) that creates a session and causes the generation of a Server Process that is either dedicated or shared.

Server Process



A Server Process is the go-between for a Client Process and the Oracle Instance.

- Dedicated Server environment – there is a single Server Process to serve each Client Process.
- Shared Server environment – a Server Process can serve several User Processes, although with some performance reduction.
- Allocation of server process in a dedicated environment versus a shared environment is covered in further detail in the *Oracle11g Database Performance Tuning* course offered by

Oracle Education.

Background Processes

As is shown here, there are both mandatory, optional, and slave background processes that are started whenever an Oracle Instance starts up. These background processes serve all system users. We will cover mandatory process in detail.

Mandatory Background Processes

- **Process Monitor Process (PMON)**
- **System Monitor Process (SMON)**
- **Database Writer Process (DBWn)**
- **Log Writer Process (LGWR)**
- **Checkpoint Process (CKPT)**
- **Manageability Monitor Processes (MMON and MMNL)**
- **Recover Process (RECO)**

Optional Processes

- **Archiver Process (ARCn)**
- **Coordinator Job Queue (CJQ0)**
- **Dispatcher (number "nnn") (Dnnn)**
- **Others**

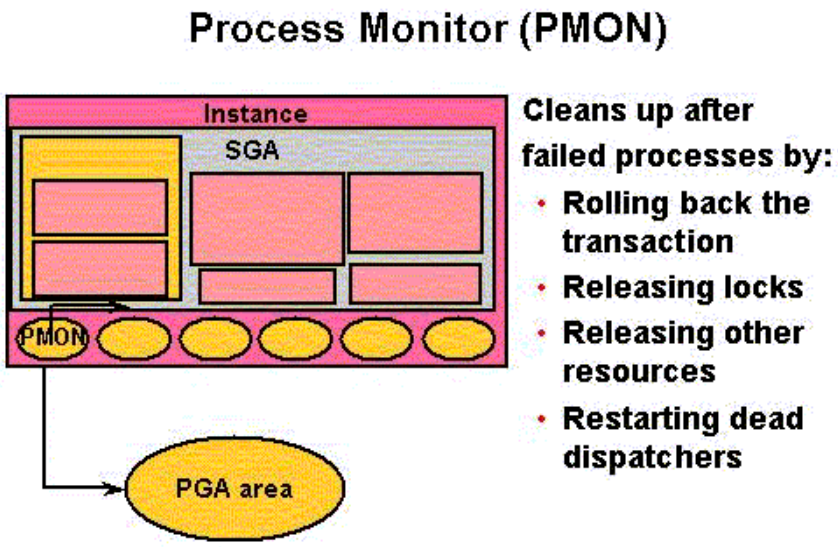
This query will display all background processes running to serve a database:

```
SELECT PNAME
FROM   V$PROCESS
WHERE  PNAME IS NOT NULL
ORDER BY PNAME;
```

PMON

The **Process Monitor (PMON)** monitors other background processes.

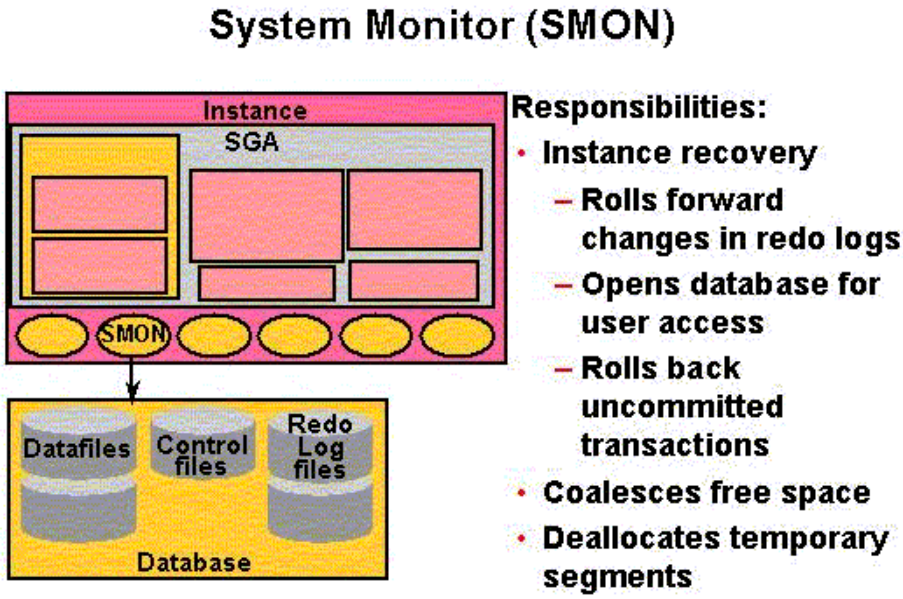
- It is a cleanup type of process that cleans up after failed processes.
- Examples include the dropping of a user connection due to a network failure or the abnormal termination (ABEND) of a user application program.
- It cleans up the database buffer cache and releases resources that were used by a failed user process.
- It does the tasks shown in the figure below.



SMON

The **System Monitor (SMON)** does system-level cleanup duties.

- It is responsible for instance recovery by applying entries in the online redo log files to the datafiles.
- Other processes can call SMON when it is needed.
- It also performs other activities as outlined in the figure shown below.



If an Oracle Instance fails, all information in memory not written to disk is lost. SMON is responsible for recovering the instance when the database is started up again. It does the following:

- Rolls forward to recover data that was recorded in a Redo Log File, but that had not yet been recorded to a datafile by DBWn. SMON reads the Redo Log Files and applies the changes to the data blocks. This recovers all transactions that were committed because these were written to the Redo Log Files prior to system failure.
- Opens the database to allow system users to logon.
- Rolls back uncommitted transactions.

SMON also does limited space management. It combines (coalesces) adjacent areas of free space in the database's datafiles for tablespaces that are dictionary managed.

It also deallocates temporary segments to create free space in the datafiles.

DBWn (also called DBWR in earlier Oracle Versions)

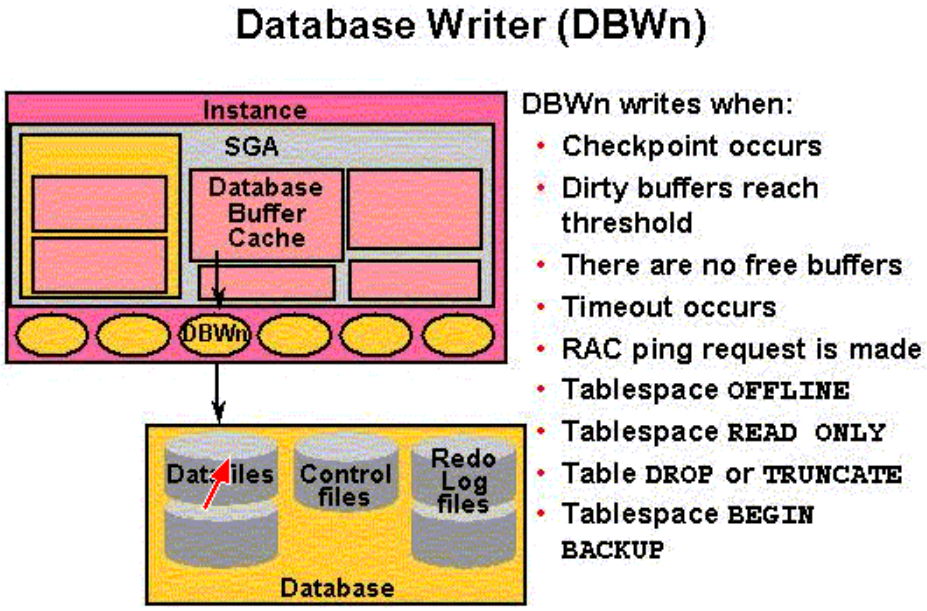
The **Database Writer** writes modified blocks from the database buffer cache to the datafiles.

- One database writer process (DBW0) is sufficient for most systems.
- A DBA can configure up to 20 DBWn processes (DBW0 through DBW9 and DBWa through DBWj) in order to improve write performance for a system that modifies data heavily.
- The initialization parameter **DB_WRITER_PROCESSES** specifies the number of DBW*n* processes.

The purpose of **DBWn** is to improve system performance by caching writes of database blocks from the **Database Buffer Cache** back to datafiles.

- Blocks that have been modified and that need to be written back to disk are termed "**dirty blocks**."
- The DBWn also ensures that there are enough free buffers in the Database Buffer Cache to service Server Processes that may be reading data from datafiles into the Database Buffer Cache.
- Performance improves because by delaying writing changed database blocks back to disk, a Server Process may find the data that is needed to meet a User Process request already residing in memory!

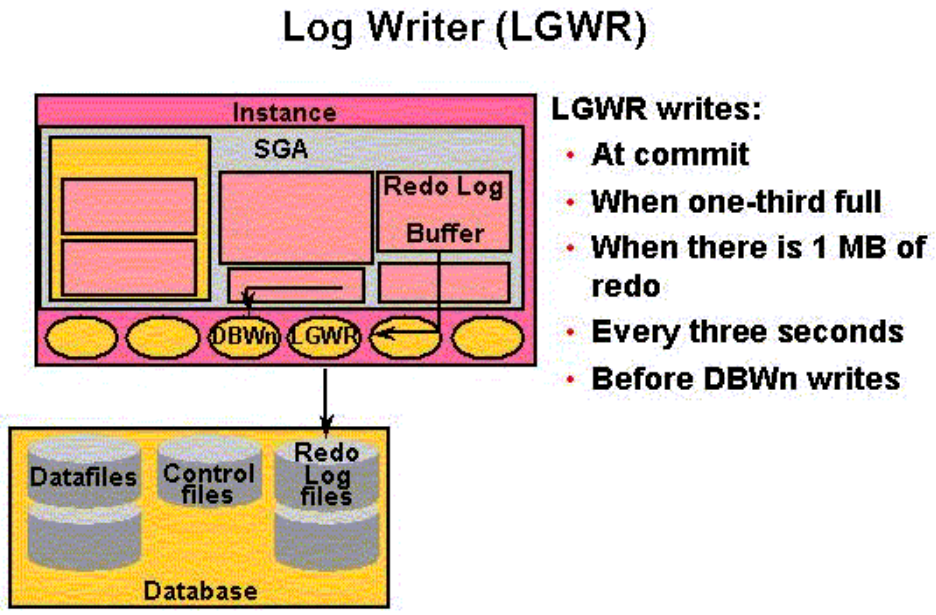
- DBWn writes to datafiles when one of these events occurs that is illustrated in the figure below.



LGWR

The **Log Writer (LGWR)** writes contents from the Redo Log Buffer to the Redo Log File that is in use.

- These are sequential writes since the Redo Log Files record database modifications based on the actual time that the modification takes place.
- LGWR actually writes before the DBWn writes and only confirms that a COMMIT operation has succeeded when the Redo Log Buffer contents are successfully written to disk.
- LGWR can also call the DBWn to write contents of the Database Buffer Cache to disk.
- The LGWR writes according to the events illustrated in the figure shown below.

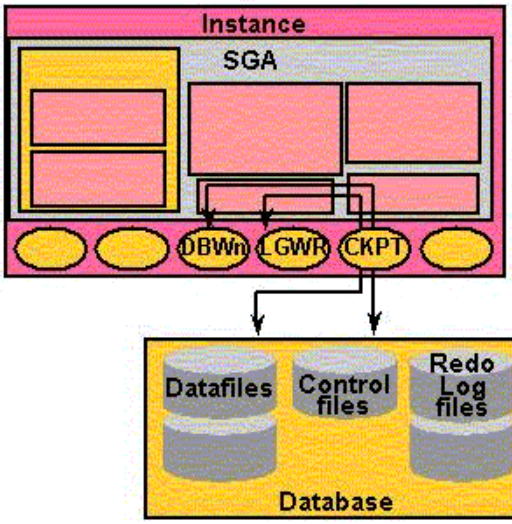


CKPT

The **Checkpoint (CPT)** process writes information to update the database control files and headers of datafiles.

- A checkpoint identifies a point in time with regard to the **Redo Log Files** where instance recovery is to begin should it be necessary.
- It can tell DBWn to write blocks to disk.
- A checkpoint is taken at a minimum, once every **three seconds**.

Checkpoint (CKPT)

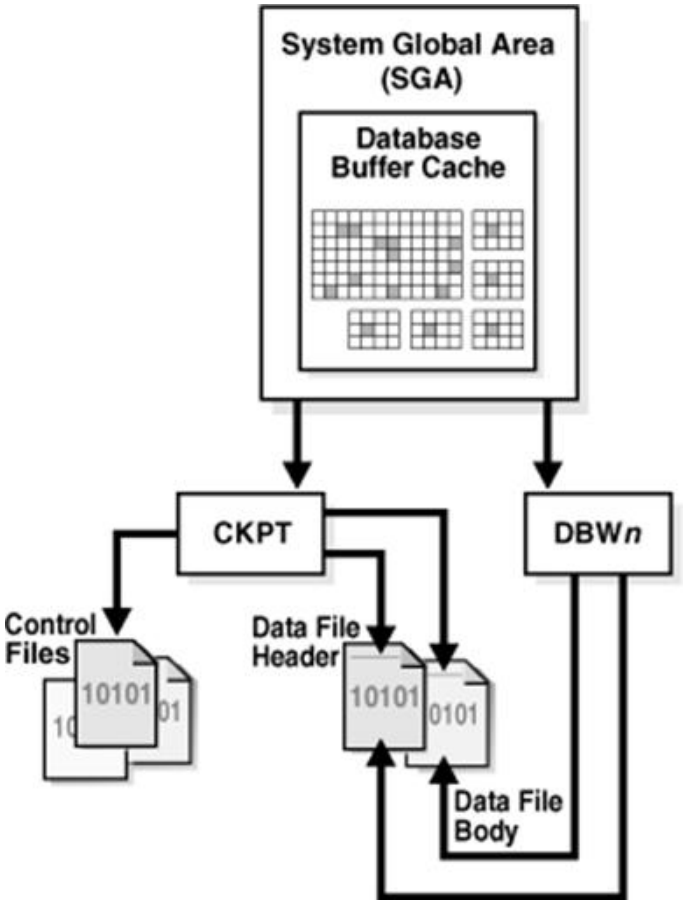


Responsible for:

- **Signaling DBWn at checkpoints**
- **Updating datafile headers with checkpoint information**
- **Updating control files with checkpoint information**

Think of a checkpoint record as a starting point for recovery. DBWn will have completed writing all buffers from the Database Buffer Cache to disk prior to the checkpoint, thus those records will not require recovery. This does the following:

- Ensures modified data blocks in memory are regularly written to disk – CKPT can call the DBWn process in order to ensure this and does so when writing a checkpoint record.
- Reduces Instance Recovery time by minimizing the amount of work needed for recovery since only Redo Log File entries processed since the last checkpoint require recovery.
- Causes all committed data to be written to datafiles during database shutdown.



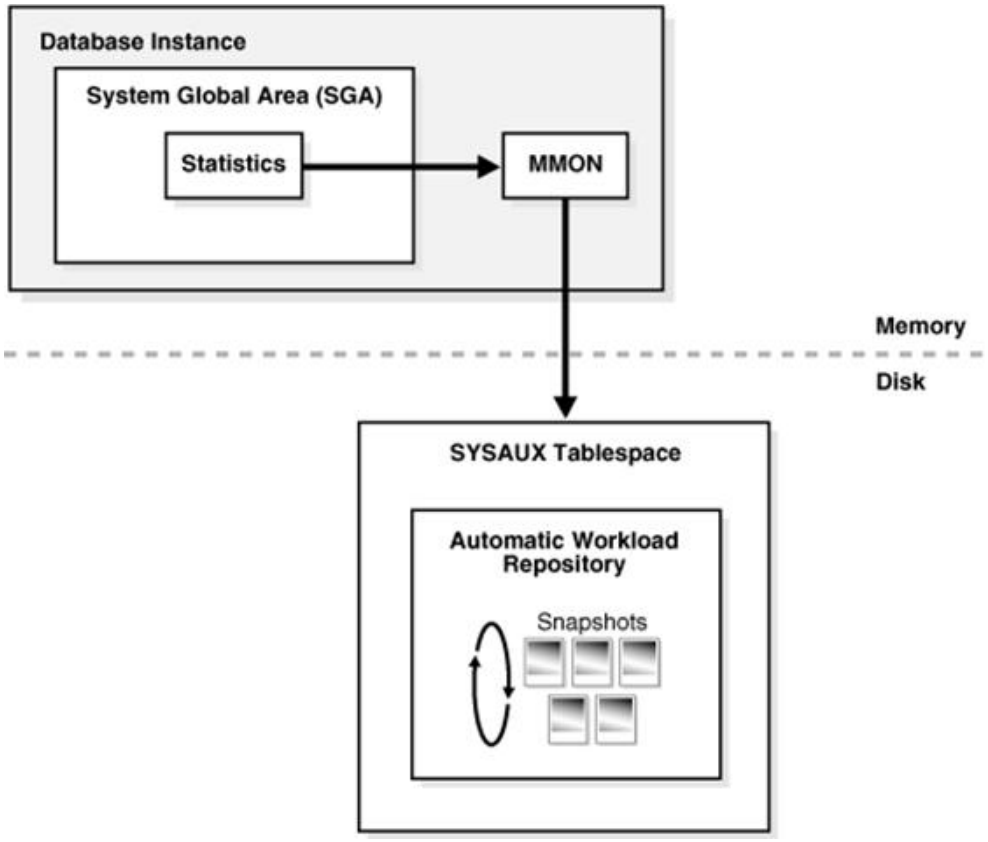
If a Redo Log File fills up and a switch is made to a new Redo Log File (this is covered in more detail in a later module), the CKPT process also writes checkpoint information into the headers of the datafiles.

Checkpoint information written to control files includes the system change number (the SCN is a number stored in the control file and in the headers of the database files that are used to ensure that all files in the system are synchronized), location of which Redo Log File is to be used for recovery, and other information.

CKPT does not write data blocks or redo blocks to disk – it calls DBWn and LGWR as necessary.

MMON and MMNL

The **Manageability Monitor Process (MMNO)** performs tasks related to the **Automatic Workload Repository (AWR)** – a repository of statistical data in the SYSAUX tablespace (see figure below) – for example, MMON writes when a **metric** violates its threshold value, taking snapshots, and capturing statistics value for recently modified SQL objects.



The **Manageability Monitor Lite Process (MMNL)** writes statistics from the Active Session History (ASH) buffer in the SGA to disk. MMNL writes to disk when the ASH buffer is full.

The information stored by these processes is used for performance tuning – we survey performance tuning in a later module.

RECO

The **Recoverer Process (RECO)** is used to resolve failures of distributed transactions in a distributed database.

- Consider a database that is distributed on two servers – one in St. Louis and one in Chicago.
- Further, the database may be distributed on servers of two different operating systems, e.g. LINUX and Windows.
- The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction.
- When RECO reestablishes a connection between the databases, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved transactions.

Optional Background Processes

Optional Background Process Definition:

- **ARCn**: Archiver – One or more archiver processes copy the online redo log files to archival storage when they are full or a log switch occurs.
- **CJQ0**: Coordinator Job Queue – This is the coordinator of job queue processes for an instance. It monitors the JOB$ table (table of jobs in the job queue) and starts job queue processes (J*nnn*) as needed to execute jobs The J*nnn* processes execute job requests created by the DBMS_JOBS package.
- **Dnnn**: Dispatcher number "nnn", for example, D000 would be the first dispatcher process – Dispatchers are optional background processes, present only when the shared server configuration is used. Shared server is discussed in your readings on the topic "Configuring Oracle for the Shared Server".
- **FBDA**: Flashback Data Archiver Process – This archives historical rows of tracked tables into Flashback Data Archives. When a transaction containing DML on a tracked table commits, this process stores the pre-image of the rows into the Flashback Data Archive. It also keeps metadata on the current rows. FBDA automatically manages the flashback data archive for space, organization, and retention

Of these, you will most often use ARCn (archiver) when you automatically archive redo log file information (covered in a later module).

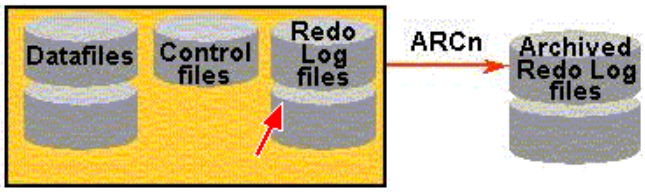
ARCn

While the **Archiver (ARCn)** is an optional background process, we cover it in more detail because it is almost always used for production systems storing mission critical information.

- The ARCn process must be used to recover from loss of a physical disk drive for systems that are "busy" with lots of transactions being completed.
- It performs the tasks listed below.

Archiver (ARCn)

- **Optional background process**
- **Automatically archives online redo logs when `ARCHIVELOG` mode is set**
- **Preserves the record of all changes made to the database**



When a Redo Log File fills up, Oracle switches to the next Redo Log File.

- The DBA creates several of these and the details of creating them are covered in a later module.
- If all Redo Log Files fill up, then Oracle switches back to the first one and uses them in a round-robin fashion by overwriting ones that have already been used.
- Overwritten Redo Log Files have information that, once overwritten, is lost forever.

ARCHIVELOG Mode:

- If ARCn is in what is termed **ARCHIVELOG** mode, then as the Redo Log Files fill up, they are individually written to Archived Redo Log Files.
- LGWR does not overwrite a Redo Log File until archiving has completed.
- Committed data is not lost forever and can be recovered in the event of a disk failure.
- Only the contents of the SGA will be lost if an Instance fails.

In **NOARCHIVELOG** Mode:

- The Redo Log Files are overwritten and **not** archived.
- Recovery can only be made to the last full backup of the database files.
- All committed transactions after the last full backup are lost, and you can see that this could cost the firm a lot of $$$.

When running in ARCHIVELOG mode, the DBA is responsible to ensure that the Archived Redo Log Files do not consume all available disk space! Usually after two complete backups are made, any Archived Redo Log Files for prior backups are deleted.

Slave Processes

Slave processes are background processes that perform work on behalf of other processes.

Innn: I/O slave processes -- simulate asynchronous I/O for systems and devices that do not support it. In **asynchronous I/O**, there is no timing requirement for transmission, enabling other processes to start before the transmission has finished.

- For example, assume that an application writes 1000 blocks to a disk on an operating system that does not support asynchronous I/O.
- Each write occurs sequentially and waits for a confirmation that the write was successful.

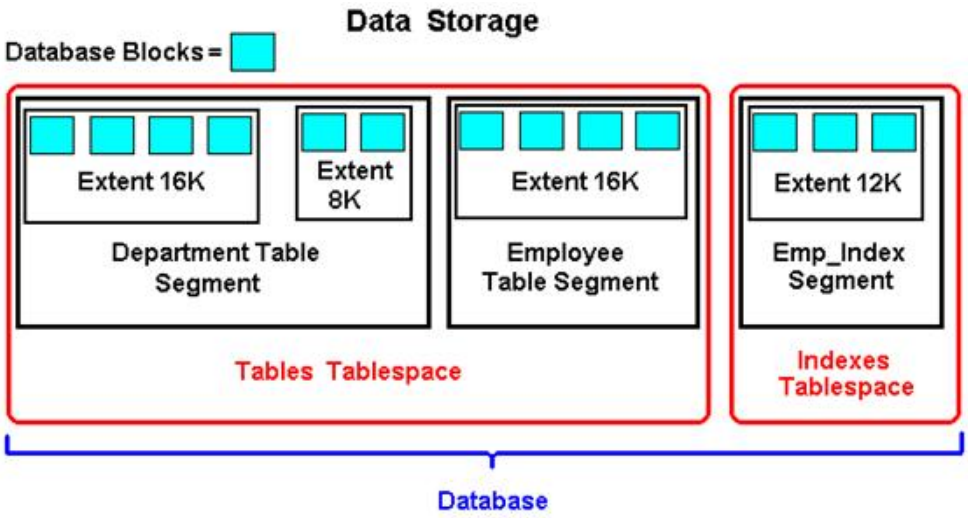
- With asynchronous disk, the application can write the blocks in bulk and perform other work while waiting for a response from the operating system that all blocks were written.

Parallel Query Slaves -- In **parallel execution** or **parallel processing**, multiple processes work together simultaneously to run a single SQL statement.

- By dividing the work among multiple processes, Oracle Database can run the statement more quickly.
- For example, four processes handle four different quarters in a year instead of one process handling all four quarters by itself.
- Parallel execution reduces response time for data-intensive operations on large databases such as data warehouses. Symmetric multiprocessing (SMP) and clustered system gain the largest performance benefits from parallel execution because statement processing can be split up among multiple CPUs. Parallel execution can also benefit certain types of **OLTP** and hybrid systems.

Logical Structure

It is helpful to understand how an Oracle database is organized in terms of a logical structure that is used to organize physical objects.



Database Blocks are allocated to extents as specified in data storage clauses. Each object (table, index, cluster) is allocated a single segment which is comprised of one or more extents. Segments are stored to Tablespaces as determined by the DBA when the object is created. All of the Tablespaces taken together comprise the Database.

Tablespace: An Oracle database must always consist of at least two **tablespaces** (**SYSTEM** and **SYSAUX**), although a typical Oracle database will multiple tablespaces.

- A tablespace is a logical storage facility (a logical container) for storing objects such as tables, indexes, sequences, clusters, and other database objects.
- Each tablespace has at least one physical datafile that actually stores the tablespace at the operating system level. A large tablespace may have more than one datafile allocated for storing objects assigned to that tablespace.
- A tablespace belongs to only one database.
- Tablespaces can be brought online and taken offline for purposes of backup and management,

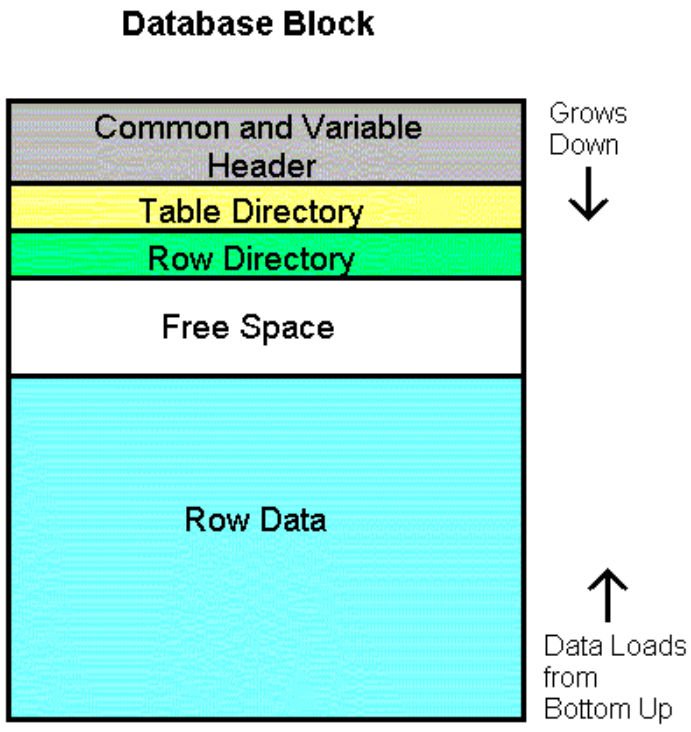
except for the **SYSTEM** tablespace that must always be online.
- Tablespaces can be in either read-only or read-write status.

Datafile: Tablespaces are stored in datafiles which are physical disk objects.
- A datafile can only store objects for a single tablespace, but a tablespace may have more than one datafile – this happens when a disk drive device fills up and a tablespace needs to be expanded, then it is expanded to a new disk drive.
- The DBA can change the size of a datafile to make it smaller or later. The file can also grow in size dynamically as the tablespace grows.

Segment: When logical storage objects are created within a tablespace, for example, an **employee table**, a **segment** is allocated to the object.
- Obviously a tablespace typically has many segments.
- A segment cannot span tablespaces but can span datafiles that belong to a single tablespace.

Extent: Each object has one segment which is a physical collection of **extents**.
- **Extents** are simply collections of **contiguous disk storage blocks**. A logical storage object such as a table or index always consists of at least one extent – ideally the initial extent allocated to an object will be large enough to store all data that is initially loaded.
- As a table or index grows, additional extents are added to the segment.
- A DBA can add extents to segments in order to tune performance of the system.
- An extent cannot span a datafile.

Block: The Oracle Server manages data at the smallest unit in what is termed a **block** or **data block**. Data are actually stored in blocks.



A **physical block** is the smallest addressable location on a disk drive for read/write operations.

An Oracle data block consists of one or more physical blocks (operating system blocks) so the data

block, if larger than an operating system block, should be an even multiple of the operating system block size, e.g., if the Linux operating system block size is 2K or 4K, then the Oracle data block should be 2K, 4K, 8K, 16K, etc in size. This optimizes I/O.

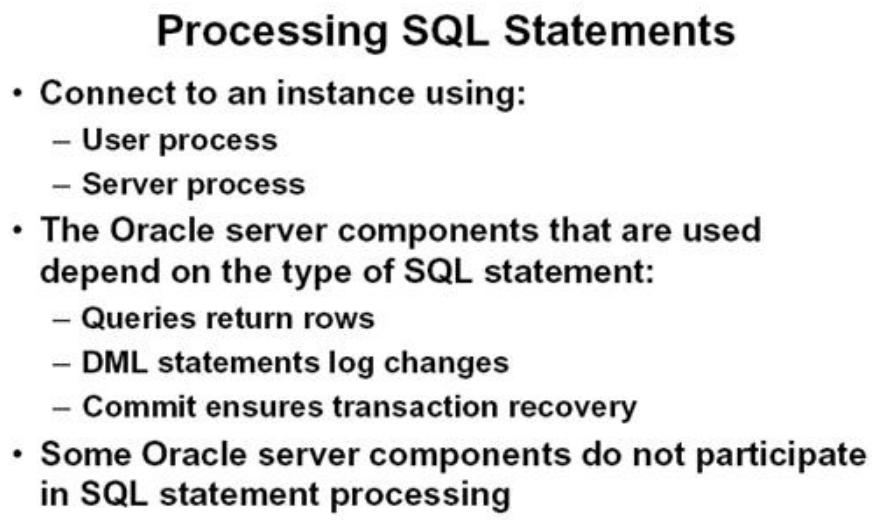
The data block size is set at the time the database is created and cannot be changed. It is set with the **DB_BLOCK_SIZE** parameter. The maximum data block size depends on the operating system.

Thus, the Oracle database architecture includes both logical and physical structures as follows:

- Physical: Control files; Redo Log Files; Datafiles; Operating System Blocks.
- Logical: Tablespaces; Segments; Extents; Data Blocks.

SQL Statement Processing

SQL Statements are processed differently depending on whether the statement is a query, data manipulation language (**DML**) to update, insert, or delete a row, or data definition language (**DDL**) to write information to the data dictionary.



Processing a query:

- Parse:
  - Search for identical statement in the Shared SQL Area.
  - Check syntax, object names, and privileges.
  - Lock objects used during parse.
  - Create and store execution plan.
- Bind: Obtains values for variables.
- Execute: Process statement.
- Fetch: Return rows to user process.

Processing a DML statement:

- Parse: Same as the parse phase used for processing a query.
- Bind: Same as the bind phase used for processing a query.
- Execute:
  - If the data and undo blocks are not already in the Database Buffer Cache, the server process reads them from the datafiles into the Database Buffer Cache.

- The server process places locks on the rows that are to be modified. The undo block is used to store the before image of the data, so that the DML statements can be rolled back if necessary.
- The data blocks record the new values of the data.
- The server process records the before image to the undo block and updates the data block. Both of these changes are made in the Database Buffer Cache. Any changed blocks in the Database Buffer Cache are marked as dirty buffers. That is, buffers that are not the same as the corresponding blocks on the disk.
- The processing of a DELETE or INSERT command uses similar steps. The before image for a DELETE contains the column values in the deleted row, and the before image of an INSERT contains the row location information.

Processing a DDL statement:

- The execution of DDL (Data Definition Language) statements differs from the execution of DML (Data Manipulation Language) statements and queries, because the success of a DDL statement requires write access to the data dictionary.
- For these statements, parsing actually includes parsing, data dictionary lookup, and execution. Transaction management, session management, and system management SQL statements are processed using the parse and execute stages. To re-execute them, simply perform another execute.

END OF NOTES