

ORACLE ARCHITECTURE FUNCTION

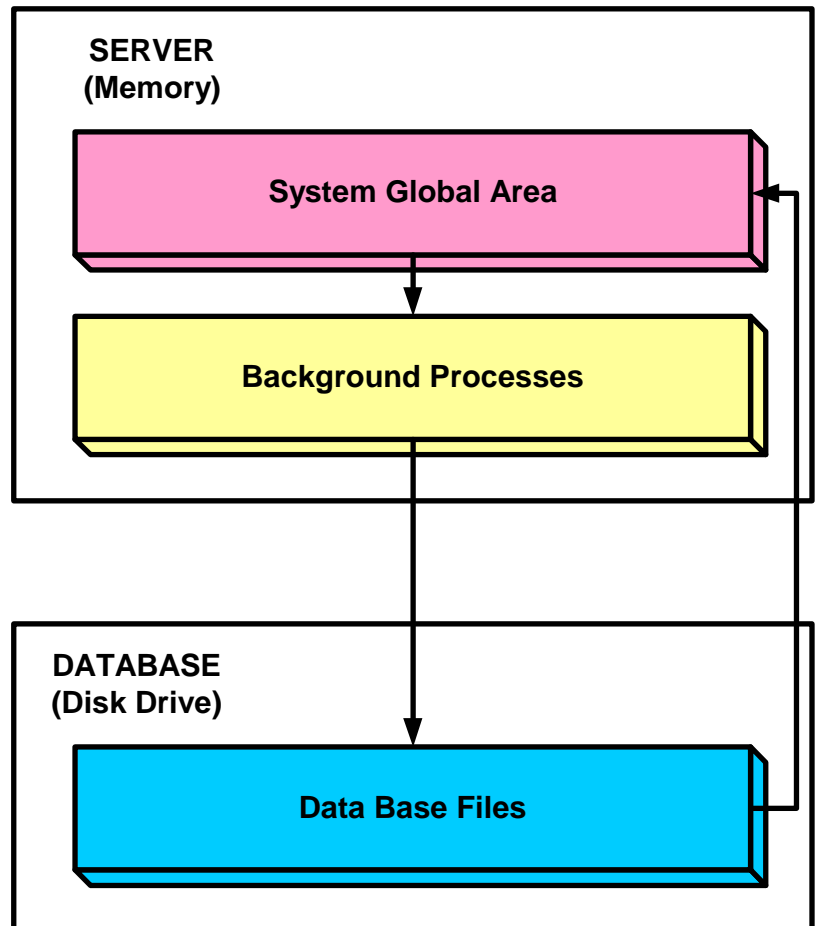
By Mark E. Donaldson

ARCHITECTURE OVERVIEW

An Oracle database consists of physical files, memory areas, background processes, and external structures. The data in the database is stored in physical files (called data files) on a disk. As it is used, that data is stored in memory. Oracle uses memory areas to improve performance and to manage the sharing of data between users. The main memory area in a database is called the System Global Area (SGA). To read and write data between the SGA and the data files, Oracle uses a set of background processes that are shared by all users.

A **database server** (also known as an **instance**) is a set of memory structures and background processes that accesses a set of database files. The relationship between servers and databases is illustrated.

The characteristics of the database server, such as the size of the SGA and the number of background processes, are specified during startup. These parameters are stored in the **init.ora** file. The name of the database is may be stored in the **init.ora** file. The **init.ora** file may in turn call a corresponding **config.ora** file. If a **config.ora** file is used, it usually only stores the parameter values for unchanging information such as database block size and the database name. The initialization files are only read during database startup, and modification to them will not take effect until the next startup.



DATABASE & EXTERNAL FILES

The database consists of the following physical files:

- Data Files
- Control Files
- Redo Log Files
- Archived Redo Log Files
- Trace Files
- Password File
- Alert File
- Initialization File
- Configuration File

REDO LOG FILES

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

Redo Log Files store all the changes made to the database. If the database is to be recovered to a point in time when it was operational, Redo Logs are used to make sure all committed transactions are committed to disk, and all uncommitted transactions are rolled back.

A database must have at least two Redo Log Files. The redo log is a set of two or more online redo log files that record all changes made to the database, **including both uncommitted and committed** changes. Redo entries are temporarily stored in redo log buffers of the system global area, and the background process LGWR writes the redo entries sequentially to an online redo log file. It also writes a commit record every time a user process commits a transaction.

The online redo log files are used in a cyclical fashion; for example, if two files constitute the online redo log, the first file is filled, the second file is filled, the first file is reused and filled, the second file is reused and filled, and so on. Each time a file is filled, it is assigned a log sequence number to identify the set of redo entries.

To avoid losing the database due to a single point of failure, Oracle can maintain multiple sets of online redo log files. A multiplexed online redo log consists of copies of online redo log files physically located on separate disks; changes made to one member of the group are made to all members. If a disk that contains an online redo log file fails, other copies are still intact and available to Oracle. System operation is not interrupted and the lost online redo log files can be easily recovered using an intact copy.

The guidelines for multiplexing are:

- The Redo Log File configuration required at least two redo log members per group, with each member on a different disk to guard against failure.
- All members of a group contain identical information and are of the same size.
- Group members are updated simultaneously.
- Each group should contain the same number of members of the same size.

You can create additional log file groups using the following SQL command:

```
ALTER DATABASE [database]
ADD LOGFILE [GROUP integer] filespec
[, [GROUP integer] filespec ]
```

To drop an entire Redo Log Group, use the following SQL command:

```
ALTER DATABASE [database]
DROP LOGFILE
{GROUP integer | ('filename' [, 'filename'] ) }
[, {GROUP integer | ('filename' ) } ]
```

At a **Redo Log Switch**, the current redo log group is assigned a log sequence number that identifies the information stored in that redo log group and is also used for synchronization. **The properties of Switches are as follows:**

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

- When LGWR stops writing to one redo log group and begins writing to another.
- When LGWR has filled one log file group.
- When a DBA can forces a log switch using the **ALTER SYSTEM SWITCH LOGFILE** command.
- A checkpoint automatically occurs at a log switch.
- Processing can continue as long as at least one member of a group is available. If a member is found to be damaged or unavailable, messages are written to the LGWR trace file and to the Alert File.

A Redo Log File can be relocated as follows:

- If the log file is current, perform a log switch:
ALTER SYSTEM SWITCH LOG FILE;
- Copy the Redo Log File from the previous location to the new location by using the operating system copy utility.
- Use the **ALTER DATABASE RENAME FILE** command to make the change in the Control Files:
**ALTER DATABASE [database]
RENAME FILE 'filename' [, 'filename']
TO 'filename' [' filename'];**
- All members of a group contain identical information.
- Group members are updated simultaneously.
- Each group should contain the same number of members of the same size.

You can add new members to existing Redo Log File Groups by using the following SQL command:

```
ALTER DATABASE [database]  
ADD LOGFILE MEMBER  
[ 'filename' [REUSE]  
[, 'filename' [REUSE] ]  
TO {GROUP integer('filename' [, 'filename'])} ];
```

You can drop a member of an online redo log group by using the following command:

```
ALTER DATABASE [database]  
DROP LOGFILE MEMBER 'filename' [, 'filename'];
```

ARCHIVE LOG FILES

Offline copies of the Redo Log Files that may be necessary to recover from media failures. When the database is set to ARCHIVE log mode, the LGWR process waits for the online redo logs files to be archived before they can be reused. If corruption is found in one online redo log file, another member from the same group is used. Archived logs are beneficial for the following reasons:

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

- A database backup, combined with archived redo log files, guarantees that all committed data can be recovered to the point of failure.
- Valid database backups can be taken while the database is online.

Optionally, filled online redo files can be archived before being reused, creating an archived redo log. Archived (offline) redo log files constitute the archived redo log. The presence or absence of an archived redo log is determined by the mode that the redo log is using:

ARCHIVELOG: The filled online redo log files are archived before they are reused in the cycle.

NOARCHIVELOG: The filled online redo log files are not archived.

In ARCHIVELOG mode, the database can be completely recovered from both instance and disk failure. The database can also be backed up while it is open and available for use. If the database's redo log operates in NOARCHIVELOG mode, the database can be completely recovered from instance failure but not from disk failure. Also, the database can be backed up only while it is completely closed.

CONTROL FILES

Contains the information needed to maintain and verify database integrity, and include the names of all the Data Files and the online and archived log files. A database must have at least one Control File. The Control File is a small binary file that describes the structure of the database. It must be available for writing by the Oracle server whenever the database is OPEN and its default name is operating system dependant. **Without this file the database cannot be mounted and recovery is difficult.** To obtain the location and names of the Control Files, use the dynamic performance view **V\$PARAMETER** or **V\$CONTROLFILE**.

The properties of Control Files are as follows:

- All necessary database files and log files are identified.
- The name of the database is stored.
- The Control File is required to MOUNT, OPEN, and maintain the database.
- Synchronization information (checkpoint & log sequence information) needed for recovery is stored.
- The recommended configuration is a minimum of two control files on different disks.
- Timestamp of database creation is stored.
- Extra backup information stored when using Recovery Manager.

To add a new Control File or change the number or location of the Control File, follow these steps:

- Shutdown the database.
- Make a copy of the existing Control File to a different device by using operating systems commands.
- Edit or add the **CONTROL_FILES** parameter and specify names for all control files.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

- Start the database.

ALERT LOG FILE

The ALERT.LOG file is written by the Oracle Server and includes error and status information for the Oracle Instance and Oracle Database. Anything that affects the database instance-wide or globally is recorded in the ALERT.LOG file.

TRACE FILES

If one of the Oracle background processes encounters an error, it will create a trace file with information about the error. The name of the background process that generated the trace file, such as PMON, SMON, LGWR, or DBWR is included as part of the trace file name. Trace files are written to the destination as specified by the parameter **BACKGROUND_DUMP_DEST**. Trace files are usually listed in the ALERT.LOG output.

INITIALIZATION (PARAMETER) FILE

The Initialization or Parameter file is used to define the characteristics and configuration parameters of an Oracle Instance (INIT<SID>.ORA). The INIT<SID>.ORA file will typically point to the actual parameter file named INIT.ORA using the keyword IFILE as follows:

```
IFILE='D:\Oracle\admin\DB1\pfile\init.ora'  
The INIT.ORA file points to Control Files.
```

CONFIGURATION FILE

The Configuration File (referred to as the CONFIG.ORA file) is an optional file that also contains parameters like the INIT.ORA file. It can only be used if the INIT.ORA file contains an Include line specifying the name of the CONFIG.ORA. This file is actually merged into the INIT.ORA file upon startup of the Instance. Used primarily to segregate a particular set of standard initialization parameters.

When you START the Instance, the Parameter Files is read. When you MOUNT the database, the Control Files are read. When you OPEN the database, the Data Files are referenced.

DATABASE LOGICAL STRUCTURES

In addition to the physical files used by the database, Oracle maintains the following logical structures:

- **Tablespaces** – The basic storage allocation in an Oracle database. Each Tablespace is composed of one or more physical (operating system) files.
- **Schemas** – Essentially the same as an account or username. Every initial Oracle database is created with the initial SYS and SYSTEM Schemas. There is no necessary relationship between a Schema and a Tablespace.
- **Segments** – Each object that takes up space is created as one or more segments. Each Segment can be in one and only one Tablespace.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

- **Extents** – Each Segment is composed of one or more extents. An Extent is a contiguous allocation of space within a Data File of a Tablespace. At the time a Segment is created, you can specify the size of the initial and next Extents, as well as the minimum and maximum number of Extents.
- **Rollback Segments** – The space Oracle writes the old value when a Table is updated, allowing users to maintain a consistent read on the Table. It also allows Oracle to restore the contents of the Table if the change is not committed.
- **Temporary Segments** – Used by Oracle during Table and Index creation and for sorting.
- **Tables** – All data in a database is stored in a table to include user data and the Data Dictionary.
- **Indexes** – Used to facilitate quick retrieval of data from the Table. Indexes are stored in separate Segments from the Table data.

This figure below illustrates the following:

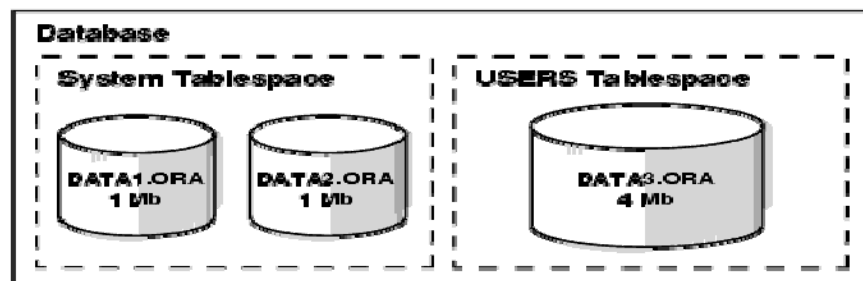
- Each database is logically divided into one or more tablespaces.
- One or more datafiles are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace.
- The combined size of a tablespace's datafiles is the total storage capacity of the tablespace (SYSTEM tablespace has 2 MB storage capacity while USERS tablespace has 4 MB).
- The combined storage capacity of a database's tablespaces is the total storage capacity of the database (6 MB).

TABLESPACES

A tablespace can be online (accessible) or offline (not accessible). A tablespace is normally online so that users can access the information within the tablespace. However, sometimes a tablespace may be taken offline to make a portion of the database unavailable while allowing normal access to the remainder of the database. This makes many administrative tasks easier to perform.

DATA BLOCKS, EXTENTS, AND SEGMENTS

Oracle allocates logical database space for all data in a database. The units of database space allocation are data blocks, extents, and segments. The following illustration shows the relationships



ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

among these data structures:

At the finest level of granularity, Oracle stores data in **data blocks** (also called logical blocks, Oracle blocks, or pages). One data block corresponds to a specific number of bytes of physical database space on disk.

The next level of logical database space is an **extent**. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information.

The level of logical database storage above an extent is called a **segment**. A segment is a set of extents, each of which has been allocated for a specific data structure, and all of which are stored in the same tablespace. For example, each table's data is stored in its own data segment, while each index's data is stored in its own index segment. If the table or index is partitioned, each partition is stored in its own segment.

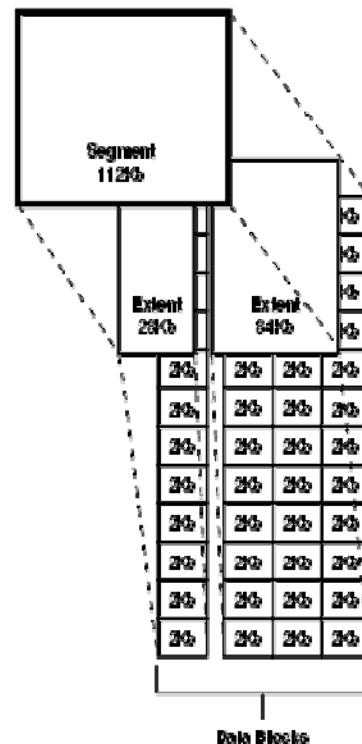
Oracle allocates space for segments in units of one extent. When the existing extents of a segment are full, Oracle allocates another extent for that segment. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk. A segment and all its extents are stored in one **tablespace**. Within a tablespace, a segment can include extents from more than one file, that is, the segment can span datafiles. However, each extent can contain data from only one datafile.

At the finest level of granularity, Oracle database data is stored in **data blocks**. One data block corresponds to a specific number of bytes of physical database space on disk. A data block size is specified for each Oracle database when the database is created. A database uses and allocates free database space in Oracle data blocks. The next level of logical database space is called an **extent**. An extent is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

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. For example, the different types of segments include:

- **Data Segment** - Each non-clustered table has a data segment. 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.



ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

- **Index Segment** - Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
- **Rollback Segment** - One or more rollback segments for a database are created by the database administrator to temporarily store undo information. The information in a rollback segment is used to generate read-consistent database information, during database recovery, and to rollback uncommitted transactions for users
- **Temporary Segment** - Temporary segments are created by Oracle 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 system for future use.

Oracle dynamically allocates space when the existing extents of a segment become full. Therefore, when the existing extents of a segment are full, Oracle allocates another extent for that segment as needed. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

SYSTEM GLOBAL AREA (SGA)

A system global area (SGA) is a group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, the data in the instance's SGA is shared among the users. Consequently, the SGA is sometimes referred to as the "shared global area". An SGA and Oracle processes constitute an Oracle instance. Oracle automatically allocates memory for an SGA when you start an instance and the operating system reclaims the memory when you shut down the instance. Each instance has its own SGA.

The SGA contains the following data structures:

- Database Buffer Cache
- Redo Log Buffer
- Shared SQL Pool
- Large Pool (optional)
- Java Pool
- Locks
- Program Global Area (PGA)

Part of the SGA contains general information about the state of the database and the instance, which the background processes need to access; this is called the fixed SGA. No user data is stored here. The SGA also includes information communicated between processes, such as locking information.

DATABASE BUFFER CACHE

The database buffer cache is the portion of the SGA that **holds copies of data blocks read from data files in the database**, such as tables, indexes, and clusters. All user processes concurrently connected to the instance share access to the database buffer cache.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

Oracle will manage the space available in the Database Buffer Cache by using a least recently used (LRU) algorithm. When free space is needed in the cache, the least recently used blocks will be written out to disk, and new data blocks will take their place in memory.

The buffers in the cache are organized in two lists:

- The **dirty list** - The dirty list holds **dirty buffers**, which contain data that has been modified but has not yet been written to disk.
- The **least recently used (LRU) list**. The least recently used (LRU) list holds free buffers, pinned buffers, and dirty buffers that have not yet been moved to the dirty list. Free buffers have not been modified and are available for use. Pinned buffers are currently being accessed.

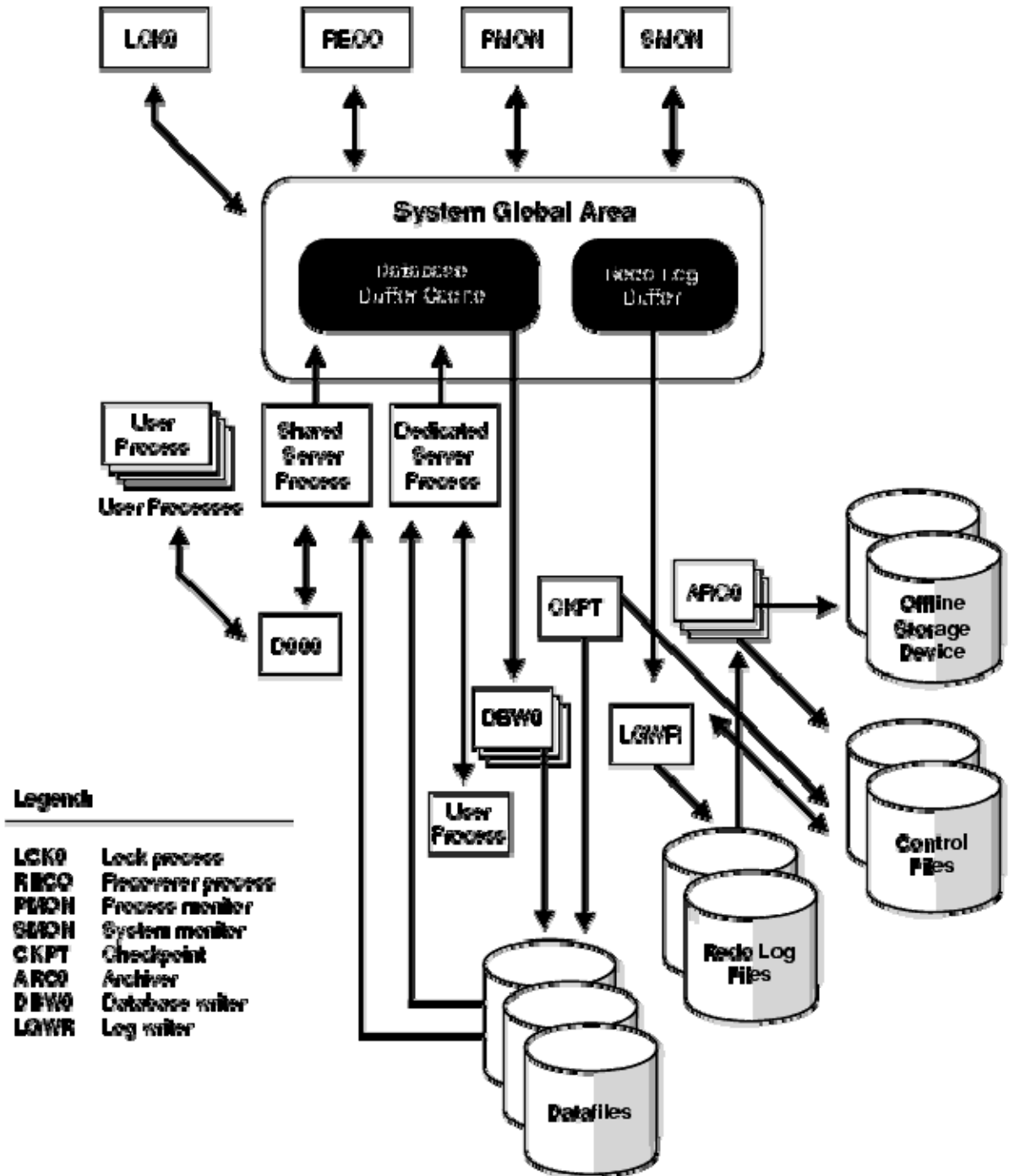
When an Oracle process accesses a buffer, the process moves the buffer to the most recently used (MRU) end of the LRU list. As more buffers are continually moved to the MRU end of the LRU list, dirty buffers age towards the LRU end of the LRU list.

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

Before reading a data block into the cache, the process must first find a free buffer. The process searches the LRU list, starting at the least recently used end of the list. The process searches either until it finds a free buffer or until it has searched the threshold limit of buffers. If the user process finds a dirty buffer as it searches the LRU list, it moves that buffer to the dirty list and continues to search. When the process finds a free buffer, it reads the data block from disk into the buffer and moves the buffer to the MRU end of the LRU list.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson



ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

If an Oracle user process searches the threshold limit of buffers without finding a free buffer, the process stops searching the LRU list and signals the DBW0 background process to write some of the dirty buffers to disk.

REDO LOG BUFFER

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 made to the database by INSERT, UPDATE, DELETE, CREATE, ALTER, or DROP operations. Redo entries are used for database recovery, if necessary.

Redo entries are copied by Oracle server processes from the user's memory space to the redo log buffer in the SGA. The redo entries take up continuous, sequential space in the buffer. The background process LGWR writes the redo log buffer to the active online redo log file (or group of files) on disk.

Every time a change is made, Oracle generates a record of both the changed and original value in the redo log buffer in memory. This record is called a redo record. Oracle records both committed and uncommitted changes in redo log buffers.

SHARED POOL

The shared pool portion of the SGA contains three major areas:

- Library Cache
- Dictionary Cache
- User Global Area (MTS Only)

DICTIONARY CACHE

Information about database objects is stored in the Data Dictionary Tables. For example, Data Dictionary information includes user account data, data filenames, segment names, extent locations, table descriptions, and privileges. **When this information is needed by the database, the data dictionary tables are read and the data that is returned is stored in the SGA Dictionary Cache. This information is for use by the Oracle server, and not user queries.** The Dictionary Cache memory space is managed by an LRU algorithm like the Database Buffer Cache.

LIBRARY CACHE

The Library Cache allows the sharing of commonly used SQL statements. The library cache includes the shared SQL areas, private SQL areas, PL/SQL procedures and packages, and control structures such as locks and library cache handles. Shared SQL areas must be available to multiple users, so the library cache is contained in the shared pool within the SGA.

Oracle represents each SQL statement it executes with a **shared SQL area** and a **private SQL area**. Oracle recognizes when two users are executing the same SQL statement and reuses the shared SQL area for those users. However, each user must have a separate copy of the statement's private SQL area.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

USER GLOBAL AREA & SHARED SQL AREAS

A shared SQL area contains the parse tree and execution plan for a single SQL statement, or for identical SQL statements. Oracle saves memory by using one shared SQL area for multiple identical DML statements, particularly when many users execute the same application. A shared SQL area is always in the shared pool.

Oracle allocates memory from the shared pool when a SQL statement is parsed; the size of this memory depends on the complexity of the statement. If a SQL statement requires a new shared SQL area and the entire shared pool has already been allocated, Oracle can deallocate items from the pool using a modified least recently used algorithm until there is enough free space for the new statement's shared SQL area. If Oracle deallocates a shared SQL area, the associated SQL statement must be reparsed and reassigned to another shared SQL area when it is next executed.

PRIVATE SQL AREAS

A private SQL area contains data such as bind information and runtime buffers. Each session that issues a SQL statement has a private SQL area. Each user that submits an identical SQL statement has his or her own private SQL area that uses a single shared SQL area; many private SQL areas can be associated with the same shared SQL area.

A private SQL area has a persistent area and a runtime area:

- The **persistent area** contains bind information that persists across executions, code for data type conversion (in case the defined data type is not the same as the data type of the selected column), and other state information (like recursive or remote cursor numbers or the state of a parallel query). The size of the persistent area depends on the number of binds and columns specified in the statement. For example, the persistent area is larger if many columns are specified in a query.
- The **runtime area** contains information used while the SQL statement is being executed. The size of the runtime area depends on the type and complexity of the SQL statement being executed and on the sizes of the rows that are processed by the statement. In general, the runtime area is somewhat smaller for INSERT, UPDATE, and DELETE statements than it is for SELECT statements, particularly when the SELECT statement requires a sort).

Oracle creates the runtime area as the first step of an execute request. For INSERT, UPDATE, and DELETE statements, Oracle frees the runtime area after the statement has been executed. For queries, Oracle frees the runtime area only after all rows are fetched or the query is canceled.

The location of a private SQL area depends on the type of connection established for a session. If a session is connected via a dedicated server, private SQL areas are located in the user's PGA. However, if a session is connected via the multi-threaded server, the persistent areas and, for SELECT statements, the runtime areas, are kept in the SGA.

ALLOCATION AND REUSE OF MEMORY IN SHARED POOL

In general, any item (shared SQL area or dictionary row) in the shared pool remains until it is flushed according to a modified LRU algorithm. The memory for items that are not being used regularly is freed if space is required for new items that must be allocated some space in the shared pool. A modified LRU algorithm allows shared pool items that are used by many sessions to remain in memory as long as they are useful, even if the process that originally created the item terminates. As

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

a result, the overhead and processing of SQL statements associated with a multi-user Oracle system is minimized.

When a SQL statement is submitted to Oracle for execution, Oracle automatically performs the following memory allocation steps:

1. Oracle checks the shared pool to see if a shared SQL area already exists for an identical statement. If so, that shared SQL area is used for the execution of the subsequent new instances of the statement. Alternatively, if there is no shared SQL area for a statement, Oracle allocates a new shared SQL area in the shared pool. In either case, the user's private SQL area is associated with the shared SQL area that contains the statement.
2. Oracle allocates a private SQL area on behalf of the session. The exact location of the private SQL area depends on the connection established for a session.

Oracle also flushes a shared SQL area from the shared pool in these circumstances:

- When the ANALYZE command is used to update or delete the statistics of a table, cluster, or index, all shared SQL areas that contain statements referencing the analyzed schema object are flushed from the shared pool. The next time a flushed statement is executed, the statement is parsed in a new shared SQL area to reflect the new statistics for the schema object.
- If a schema object is referenced in a SQL statement and that object is later modified in any way, the shared SQL area is *invalidated* (marked invalid) and the statement must be reparsed the next time it is executed. If you change a database's global database name, *all* information is flushed from the shared pool.
- The administrator can manually flush all information in the shared pool to assess the performance (with respect to the shared pool, not the data buffer cache) that can be expected after instance startup without shutting down the current instance.

JAVA POOL

The Java Pool services the parsing requirements for Java commands. The Java Pool's size is set, in bytes, with the **JAVA_POOL_SIZE** init.ora parameter.

LARGE POOL

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

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

By allocating session memory from the large pool for the multi-threaded server or for Oracle XA, Oracle can use the shared pool primarily for caching shared SQL and avoid the performance overhead caused by shrinking the shared SQL cache.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

The Large Pool is used to allocate sequential IO buffers from shared memory. Recovery Manager (RMAN) uses the Large Pool for backup and restore when you set the **BACKUP_DISK_IO_SLAVES** or **BACKUP_TAPE_IO_SLAVES** parameters. You can also increase archiving performance by using the **ARCH_IO_SLAVES** parameter. If the **LARGE_POOL_SIZE** initialization parameter is not set, then Oracle attempts to allocate shared memory buffers from the shared pool in the SGA. **If the **LARGE_POOL_SIZE** is set but is not large enough, the allocation fails and the Oracle component requesting the buffers behaves as follows:**

- Log archiving fails and returns an error.
- The RMAN command writes a message to the Alert File and does not use IO slaves for that operation.

BACKGROUND PROCESSES

The Background Processes perform common functions that are needed to service the requests from several concurrent users. Each Oracle Instance may use several background processes depending on the configuration.

Default Processes:

- **Database Writer (DBWR)** – Responsible for writing changed data to the database. The database writer writes modified blocks from the database buffer cache to the datafiles. Although one database writer process (DBW0) is sufficient for most systems, you can configure additional processes (DBW1 through DBW9) to improve write performance for a system that modifies data heavily. The initialization parameter **DB_WRITER_PROCESSES** specifies the number of DBWn processes.
- **Log Writer (LGWR)** – Records changes registered in the Redo Log Buffer to the database.
- **System Monitor (SMON)** – Primary function is to check for consistency and initiate recovery of the database when the database is opened.
- **Process Monitor (PMON)** – Cleans up the resources if one of the processes fails.
- **Checkpoint Process (CKPT)** – Responsible for updating the database status information whenever changes in the Database Buffer Cache are permanently recorded in the database.

Additional Processes:

- **Archiver (ARCH)** – copies the contents of an online Redo Log File to another location (disk file) when that log becomes full.
- **Shared Server Process (S)** – created when using the Multi-threaded Server (MTS), these processes allow many user processes to create connections to an Oracle Instance.
- **Dispatcher (D)** – responsible for receiving connection requests from the listener and directing each request to the least busy Shared Server process when the MTS is being used.
- **Recovery (RECO)** – used by the Oracle distributed transaction facility to recover from failures involving distributed transactions.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

- **Lock (LCK)** – used by Oracle running with the Parallel Server Option to provide inter-instance locking.
- **Parallel Query (P)** - used by Oracle running with the Parallel Server Option and are responsible for executing SQL statements in parallel. The number is determined by the **PARALLEL_MIN_SERVERS** parameter in the INIT.ORA file.
- **Queue Monitor (QMN)** – used by the Oracle Advanced Queuing Option to monitor message queues.
- **Job Queue (JNP)** – used when snapshots are implemented in an Oracle database running the Distributed Option.

DATABASE WRITER (DBWR)

The server process records changes to rollback and data blocks in the buffer cache. The Database Writer (DBWR) writes the dirty buffers from the Database Buffer Cache to the data files. The DBWR defers writing to the Data Files until one of the following events occurs:

- The number of dirty buffers reaches a threshold value.
- A process scans a specified number of blocks when scanning for free buffers and cannot find any.
- A timeout occurs (Every three seconds).
- Triggered by an event such as closing the database.

Since Oracle uses write-ahead logging, DBWn does not need to write blocks when a transaction commits. Instead, DBWn is designed to perform batched writes with high efficiency. In the most common case, DBWn writes only when more data needs to be read into the system global area and too few database buffers are free. The least recently used data is written to the datafiles first. DBWn also performs writes for other functions such as checkpointing.

LOG WRITER (LGWR)

The Log Writer (LGWR) is a background process that writes entries from the Redo Log Buffer into the Redo Log Files. The LGWR performs sequential writes to the Redo Log File under the following situations:

- When the Redo Log Buffer is 1/3 full.
- When a timeout occurs (every three seconds) **LOG_CHECKPOINT_TIMEOUT**.
- Before DBWR writes modified blocks in the database buffer cache to the data files.
- When a transaction commits. When a transaction commits, the Oracle server places a commit record along with the System Change Number (SCN) in the Redo Log Buffer.

ROLLBACK SEGMENTS

Before making a change. The server process saves the old value into a rollback segment. This image is used to (see side figure):

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

- Undo the changes if the transactions is rolled back.
- Ensure that other transactions do not see uncommitted changes made by the DML statement (read consistency).
- Recover the database to a consistent state in case of failures.
- Rollback segments, like tables and indexes, exist in data files and parts of them are brought into the database buffer cache when required.

Oracle can roll back multiple transactions simultaneously as needed. All transactions system-wide that were active at the time of failure are marked as DEAD. Instead of waiting for SMON to roll back dead transactions, new transactions can recover blocking transactions themselves to get the row locks they need.

ROLLING FORWARD & ROLLING BACK

Database buffers in the buffer cache in the SGA are written to disk only when necessary, using a least-recently-used algorithm. Because of the way that the DBWn process uses this algorithm to write database buffers to data files, data files might contain some data blocks modified by uncommitted transactions and some data blocks missing changes from committed transactions.

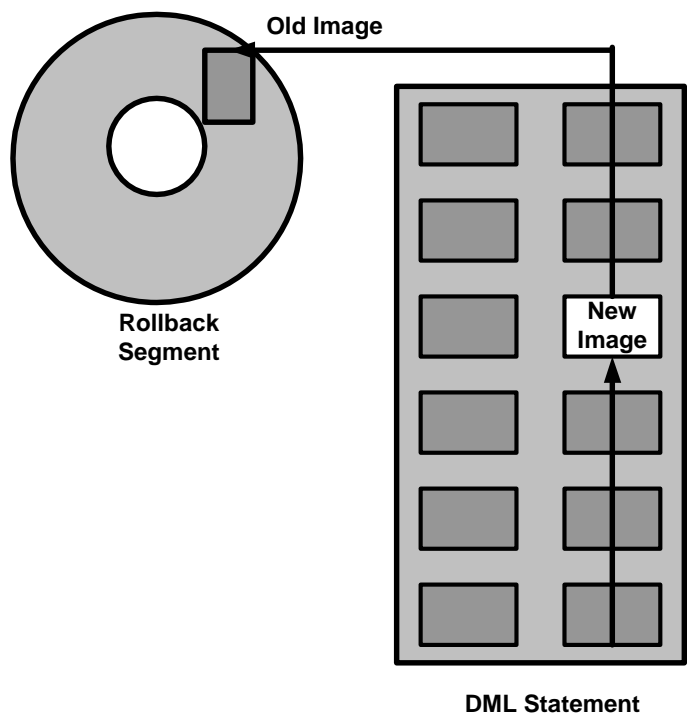
Two potential problems can result if an instance failure occurs:

- Data blocks modified by a transaction might not be written to the data files at commit time and might only appear in the redo log. Therefore, the redo log contains changes that must be reapplied to the database during recovery.
- After the roll forward phase, the data files may contain changes that had not been committed at the time of the failure. These uncommitted changes must be rolled back to ensure transactional consistency. These changes were either saved to the data files before the failure, or introduced during the roll forward phase.

To solve this dilemma, two separate steps are generally used by Oracle for a successful recovery of a system failure: rolling forward with the redo log (cache recovery) and rolling back with the rollback segments (transaction recovery).

REDO LOG & ROLLING FORWARD

ROLLBACK SEGMENT

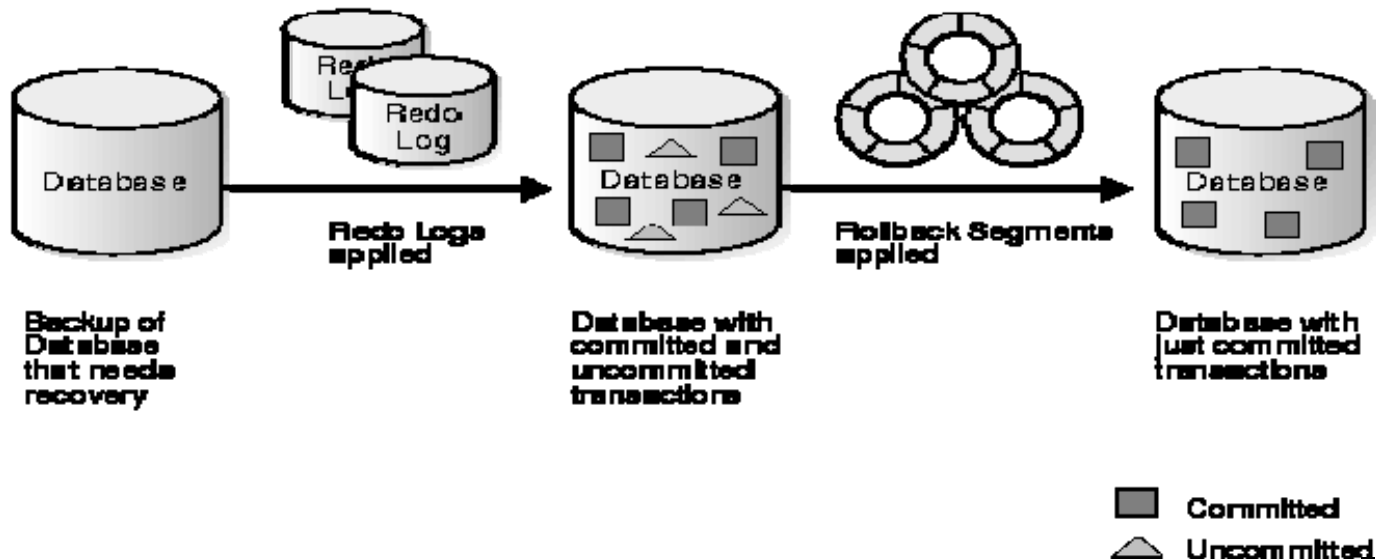


ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

The redo log is a set of operating system files that record all changes made to any database buffer, including data, index, and rollback segments, **whether the changes are committed or uncommitted**. Each redo entry is a group of change vectors describing a single atomic change to the database. The redo log protects changes made to database buffers in memory that have not been written to the data files.

The first step of recovery from an instance or disk failure is to **roll forward**, or reapply all of the



changes recorded in the redo log to the data files. Because rollback data is also recorded in the redo log, rolling forward also regenerates the corresponding rollback segments. This is called cache recovery.

Rolling forward proceeds through as many redo log files as necessary to bring the database forward in time. Rolling forward usually includes online redo log files and may include archived redo log files.

After roll forward, the data blocks contain all committed changes. They may also contain uncommitted changes that were either saved to the data files before the failure, or were recorded in the redo log and introduced during roll forward.

ROLLBACK SEGMENTS & ROLLING BACK

Rollback segments record database actions that should be undone during certain database operations. In database recovery, rollback segments undo the effects of uncommitted transactions previously applied by the rolling forward phase.

After the roll forward, any changes that were not committed must be undone. After redo log files have reapplied all changes made to the database, then the corresponding rollback segments are used. Rollback segments are used to identify and undo transactions that were never committed, yet were either saved to the data files before the failure, or were applied to the database during the roll forward. This process is called **rolling back** or **transaction recovery**.

The figure above illustrates rolling forward and rolling back, the two steps necessary to recover from any type of system failure.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

CHECKPOINT PROCESS

Database checkpoints ensure that all modified data file buffers are written to the database files. The database header files are then marked current, and the checkpoint sequence number is recorded in the Control File. The checkpoint event means that all modified or dirty buffers in the Database Buffer Cache and the Log Buffer have been written to disk since the last checkpoint. This is one event that synchronizes the write operation between LGWR and DBWR. More frequent checkpoints reduce the time necessary for recovering from instances failure at the possible expense of performance.

Checkpoints occur as follows:

- At every log switch.
- A specified number of seconds after the last database checkpoint (set using **LOG_CHECKPOINT_TIMEOUT**).
- When a number of OS blocks have been written to the Redo Log Files since the last Checkpoint (set using **LOG_CHECKPOINT_INTERVAL**).
- At Instance shutdown, unless Instance is aborted.
- When forced by a DBA (**ALTER SYSTEM CHECKPOINT** command).
- When a tablespace is taken offline or an online backup is started.

Synchronization occurs as follows:

- At each checkpoint, the checkpoint number is updated in every database file header and in the Control File.
- The checkpoint number acts as a synchronization marker for redo, control, and data files. If they have the same checkpoint number, the database is considered to be in a consistent state.
- The Control File confirms that all files are at the same checkpoint number during database startup. Any inconsistency between the checkpoint numbers in the various header files results in failure, and the database cannot be opened. Recovery is required.

Checkpoints expedite instance recovery because at every checkpoint all changed data is written to disk. **Once data resides in data files, redo log entries before the last checkpoint need not be applied again during roll forward phase of instance recovery.** The initialization parameter **LOG_CHECKPOINTS_TO_ALERT** can be used to determine if checkpoints are occurring at the desired frequency.

You can use three initialization parameters to influence how aggressively Oracle advances the checkpoint as shown in the table below:

Table 25-1 Initialization Parameters Influencing Checkpoints

Parameter	Purpose
LOG_CHECKPOINT_TIMEOUT	Limit the number of seconds between the most recent redo

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

	record and the checkpoint.
LOG_CHECKPOINT_INTERVAL	Limit the number of redo records between the most recent redo record and the checkpoint.
FAST_START_IO_TARGET	Limit instance recovery time by controlling the number of data blocks Oracle processes during instance recovery.

LOG_CHECKPOINT_TIMEOUT

Set the initialization parameter **LOG_CHECKPOINT_TIMEOUT** to a value n (where n is an integer) to require that the latest checkpoint position follow the most recent redo block by no more than n seconds. In other words, at most, n seconds' worth of logging activity can occur between the most recent checkpoint position and the end of the redo log. This forces the checkpoint position to keep pace with the most recent redo block

You can also interpret **LOG_CHECKPOINT_TIMEOUT** as specifying an upper bound on the time a buffer can be dirty in the cache before DBWn must write it to disk. For example, if you set **LOG_CHECKPOINT_TIMEOUT** to 60, then no buffers remain dirty in the cache for more than 60 seconds. The default value for **LOG_CHECKPOINT_TIMEOUT** is 1800.

LOG_CHECKPOINT_INTERVAL

Set the initialization parameter **LOG_CHECKPOINT_INTERVAL** to a value n (where n is an integer) to require that the checkpoint position never follow the most recent redo block by more than n blocks. In other words, at most n redo blocks can exist between the checkpoint position and the last block written to the redo log. In effect, you are limiting the amount of redo blocks that can exist between the checkpoint and the end of the log.

Oracle limits the maximum value of **LOG_CHECKPOINT_INTERVAL** to 90% of the smallest log to ensure that the checkpoint advances far enough to eliminate "log wrap". Log wrap occurs when Oracle fills the last available redo log file and cannot write to any other log file because the checkpoint has not advanced far enough. By ensuring that the checkpoint never gets too far from the end of the log, Oracle never has to wait for the checkpoint to advance before it can switch logs.

LOG_CHECKPOINT_INTERVAL is specified in redo blocks. Redo blocks are the same size as operating system blocks. Use the **LOG_FILE_SIZE_REDO_BLKs** column in **V\$INSTANCE_RECOVERY** to see the number of redo blocks corresponding to 90% of the size of the smallest log file.

FAST_START_IO_TARGET

You can only use the initialization parameter **FAST_START_IO_TARGET** if you have the Oracle8i Enterprise Edition. You can set this parameter to n, where n is an integer limiting to n the number of buffers that Oracle processes during crash or instance recovery. Because the number of I/Os to be processed during recovery correlates closely to the duration of recovery, the **FAST_START_IO_TARGET** parameter gives you the most precise control over the duration of recovery.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

FAST_START_IO_TARGET advances the checkpoint because DBWn uses the value of **FAST_START_IO_TARGET** to determine how much writing to do. Assuming that users are making many updates to the database, a low value for this parameter forces DBWn to write changed buffers to disk. The CKPT process reflects this progress as the checkpoint advances. Of course, if user activity is low or non-existent, DBWn does not have any buffers to write, so the checkpoint does not advance.

The smaller the value of **FAST_START_IO_TARGET**, the better the recovery performance, since fewer blocks require recovery. If you use smaller values for this parameter, however, you impose higher overhead during normal processing, since DBWn must write more buffers to disk more frequently.

DATABASE FILE SYNCHRONIZATION

An Oracle database cannot be OPENED if all datafiles, redo logs, and control files are not synchronized. In this case, recovery is required.

- **For the database to OPEN, all datafiles must have the same checkpoint number** unless they are offline or part of a “read only” tablespace.
- **Synchronization of all Oracle files is based on the current redo log checkpoint and sequence number.**
- Archived and redo logs files recover committed transactions and rollback uncommitted transactions to synchronize the database files.
- Archived and redo **log files are automatically requested by Oracle during the recovery phase.** Make sure logs exist in the requested location.

ARCHIVE BACKGROUND PROCESS (ARCH)

The ARCH process is an optional process. When enabled, it archives the Redo Log Files to the designated storage area. The ARCH process initiates when a log switch occurs and copies one member of the last un-archived Redo Log group to the destination specified by the INIT.ORA parameter **LOG_ARCHIVE_DEST**. When the database is set in ArchiveLog mode, the LGWR process checks if a Redo Log File has been archived before it re-uses the log file.

INSTANCE RECOVERY

To recover from an Instance failure start the Instance using the STARTUP command. The Oracle Server will automatically recover, performing both roll-forward and rollback phases.

ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

INSTANCE RECOVERY PHASES	
Step	Explanation
1	Unsyncronized Files By reading the Control File SCN's, Oracle determines if a database needs recovery when unsyncronized files are found. Instance failure can cause this to happen. This situation causes loss of uncommitted data since memory is not written to disk and files are not syncronized before shutdown.
2	Roll-Forward Process DBWR writes both committed and uncommitted data to the datafiles. The purpose of the roll-forward process is to apply all changes recorded in the redo log files since the last checkpoint to the data blocks. <ul style="list-style-type: none">• Rollback segments are populated during the roll-forward phase. Since redo logs store both before and after data images, a rollback segment entry is added if an uncommitted block is found in the datafile and no rollback entry exists.• Redo logs are applied using log buffers. The buffers used are marked for recovery and do not participate in normal transactions until they are relinquished by recovery process.• Redo logs are only applied to a read-only datafile if there is a status conflict.
3	Committed & Uncommitted Data in Datafiles Once the roll-forward phase has successfully completed, all committed data resides in the datafiles though uncommitted data still might exist.
4	Roll-Back Phase To remove the uncommitted data from the files, rollback segments populated during the roll-forward phase are used. Blocks are rolled back when requested by either the Oracle server or a user, depending on who requests the block first.
5	Committed Data in Datafiles When both the roll-forward and rollback phases have completed, only committed data resides on disk.
6	Syncronized Data Files All datafiles are now syncronized.

PROCESSING A DML STATEMENT

A data manipulation language (DML) statement requires two phases of processing:

- Parse which is similar to the parse phrase used for processing a query.
- Execute

Consider an example in which a user executes an update command of the following form:

UPDATE emp SET sal=sal * 1 WHERE empno=7369

What follows are the steps used in executing the UPDATE statement (see diagram below):

1. The server process reads the data and rollback blocks from the data files, if they are not already in the Buffer Cache.
2. Copies of the blocks that are read are placed in the Buffer Cache.

ORACLE ARCHITECTURE FUNCTION

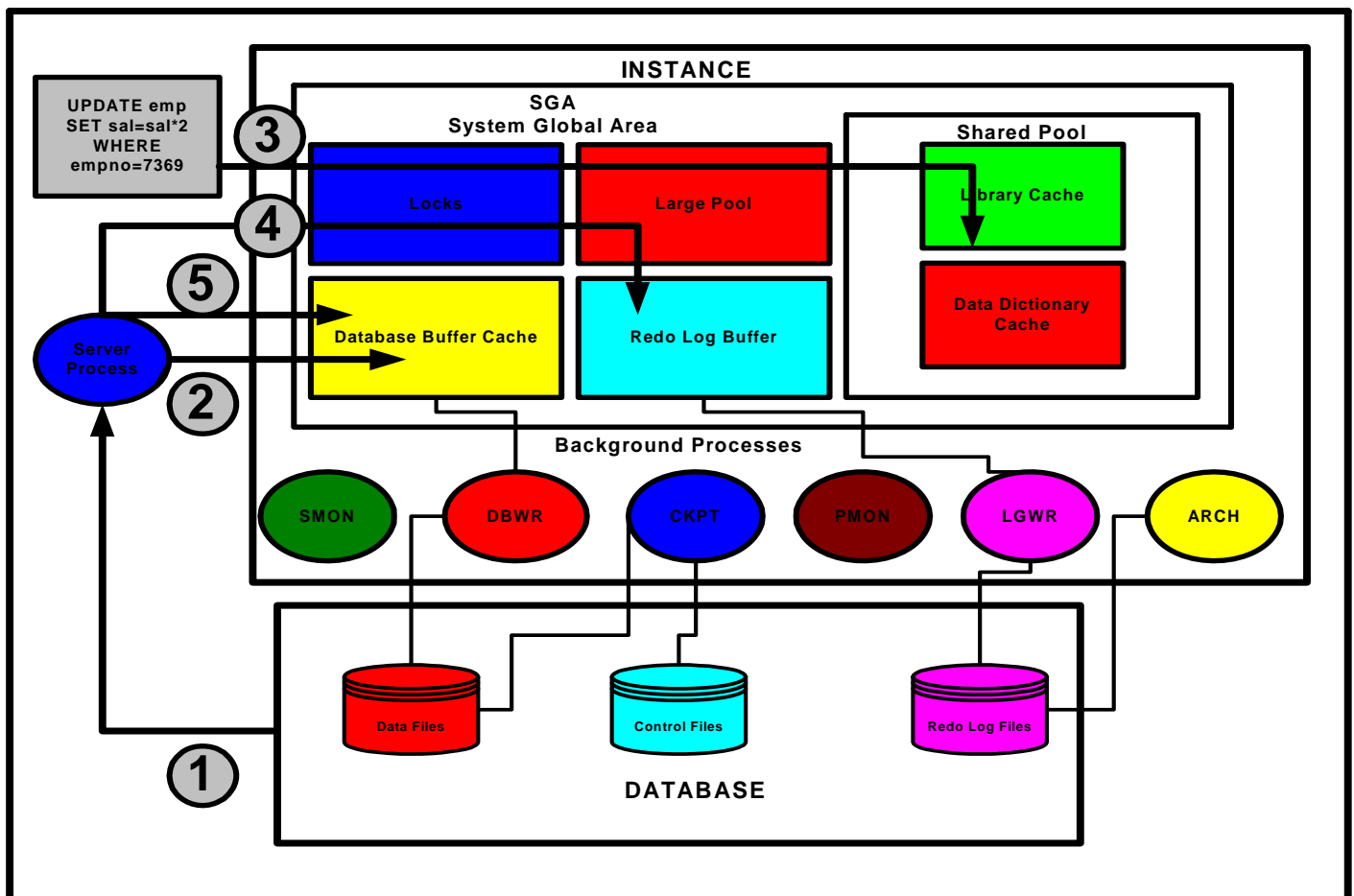
By Mark E. Donaldson

3. The server process places locks on the data.
4. The server process records the changes to be made to the rollback (before image) and to the data (new image) in the Redo Log Buffer.
5. The server process records the before image to the rollback block and updates the data block, both in the Database Buffer Cache. Both the 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

COMMIT PROCESSING

Whenever a transaction commits, the Oracle server assigns a commit System Change number (SCN) to the transaction, which is monotonically incremented and is unique within the database. The SCN is used by the Oracle server as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. Using the SCN allows the Oracle server to perform consistency checks without depending on the date and time of the operating system.

When a COMMIT is issued, the following steps occur (see diagram below):



ORACLE ARCHITECTURE FUNCTION

By Mark E. Donaldson

1. The server process places a commit record, along with the SCN, in the Redo Log Buffer.
2. Log Writer (LGWR) performs a contiguous write of all Redo Log Buffer entries up to and including the commit record to the Redo Log Files. After this point, the Oracle server can guarantee that the changes will not be lost even in case of failures.
3. The user is informed that the COMMIT is complete.
4. The server process records information to indicate that the transaction is complete and that resource locks can be released.

Notes:

- Flushing of the dirty buffers to the data file is performed independently by DBWR, and can take place either before or after the commit.
- Rolling back a transaction does not trigger LGWR to write to disk. The Oracle server always rolls back uncommitted changes when recovering from failures. If there is a failure after a rollback, before the rollback entries are recorded on disk, the absence of a commit record is sufficient to ensure that the changes made by the transaction are rolled back.

