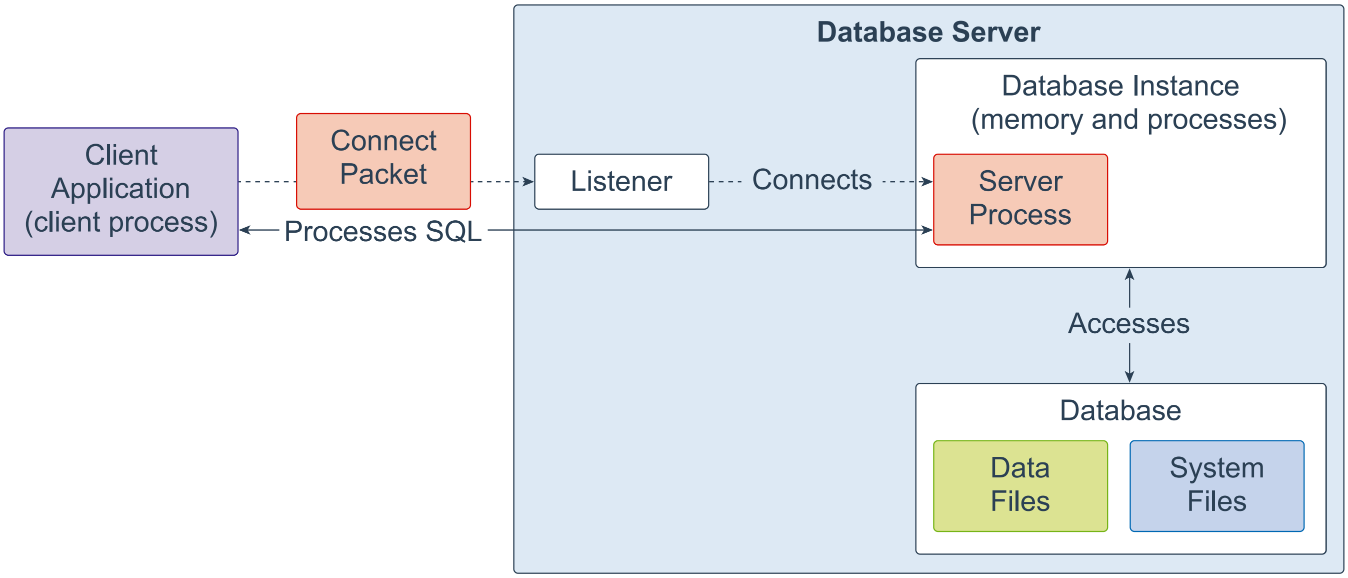
Oracle Database 21c Technical Architecture

Database Server



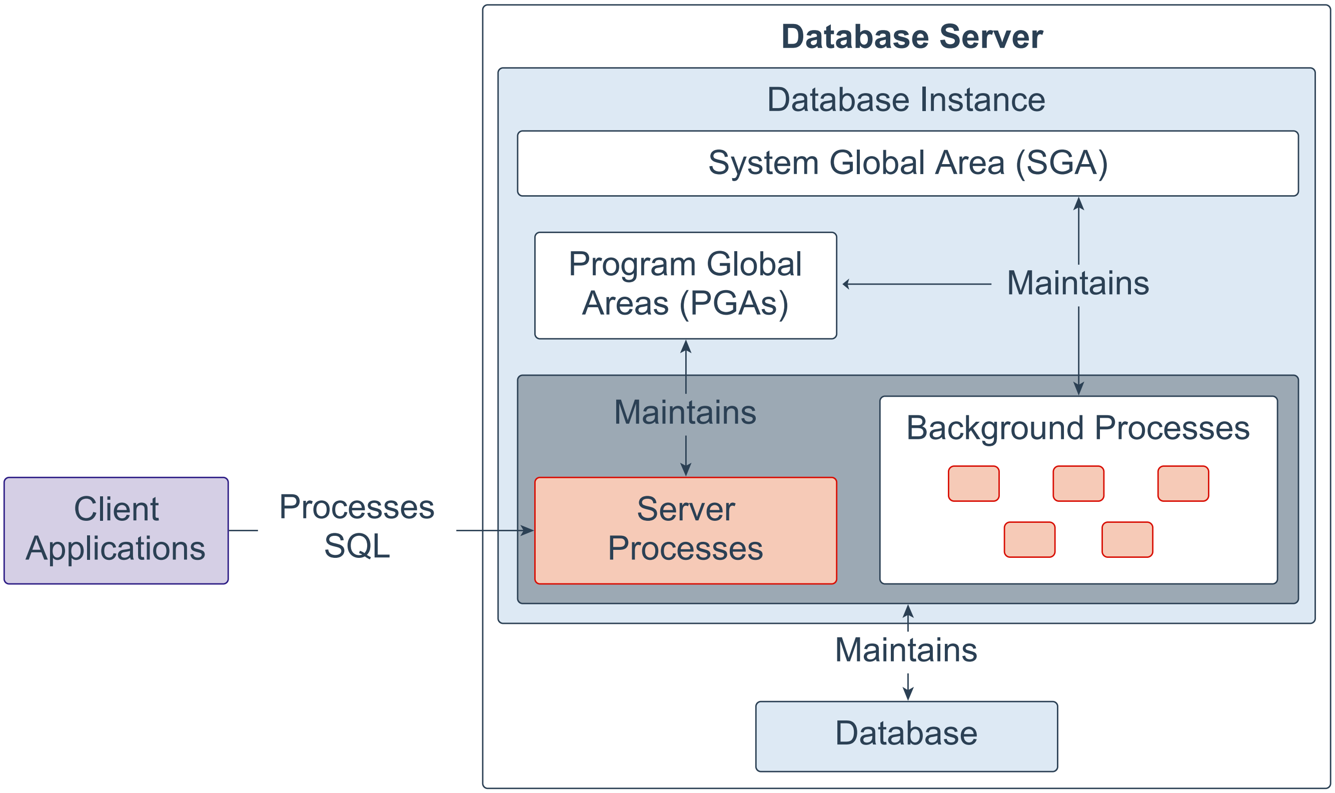
An Oracle Database consists of at least one database instance and one database. The database instance handles memory and processes. The multitenant container database consists of physical files called data files. An Oracle Database also uses several database system files during its operation.

A single-instance database architecture consists of one database instance and one database. A one-to-one relationship exists between the database and the database instance. Multiple single-instance databases can be installed on the same server machine. There are separate database instances for each database. This configuration is useful to run different versions of Oracle Database on the same machine.

An Oracle Real Application Clusters (Oracle RAC) database architecture consists of multiple instances that run on separate server machines. All of them share the same database. The cluster of server machines appear as a single server on one end, and end users and applications on the other end. This configuration is designed for high availability, scalability, and high-end performance.

The listener is a database server process. It receives client requests, establishes a connection to the database instance, and then hands over the client connection to the server process. The listener can run locally on the database server or run remotely. Typical Oracle RAC environments are run remotely.

Database Instance

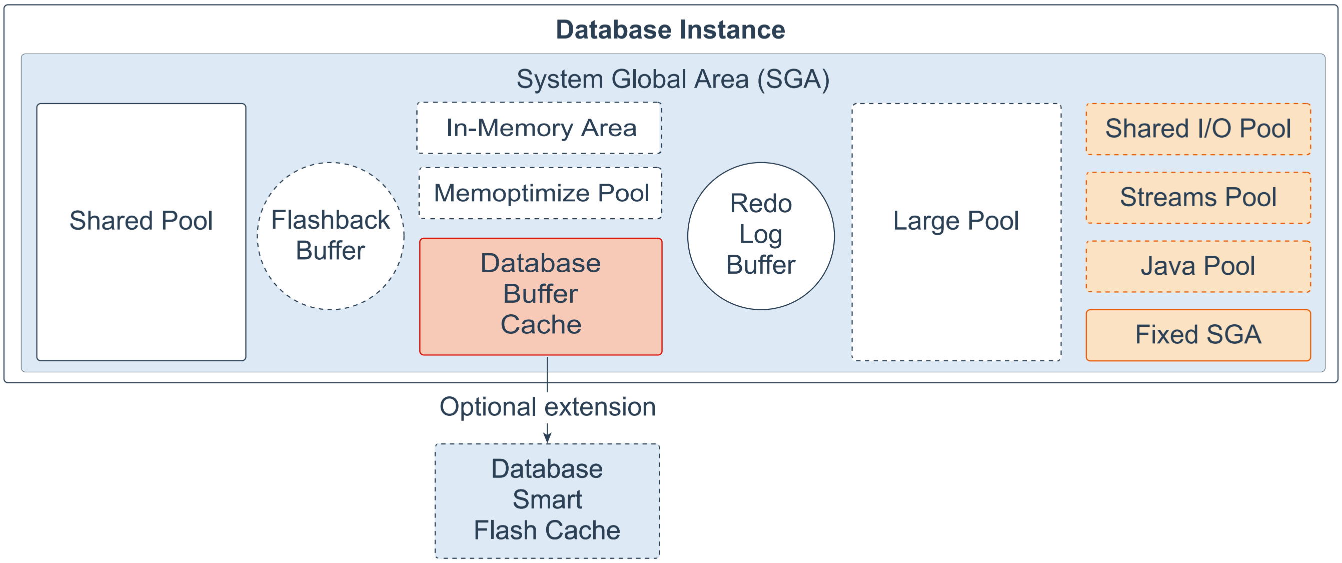


A database instance contains a set of Oracle Database background processes and memory structures. The main memory structures are the System Global Area (SGA) and the Program Global Areas (PGAs). The background processes operate on the stored data (data files) in the database and use the memory structures to do their work. A database instance exists only in memory.

Oracle Database also creates server processes to handle the connections to the database on behalf of client programs, and to perform the work for the client programs; for example, parsing and running SQL statements, and retrieving and returning results to the client programs. These types of server processes are also referred to as foreground processes.

For more information, see Oracle Database Instance.

System Global Area



The System Global Area (SGA) is the memory area that contains data and control information for one Oracle Database instance. All server and background processes share the SGA. When you start a database instance, the amount of memory allocated for the SGA is displayed. The SGA includes the following data structures:

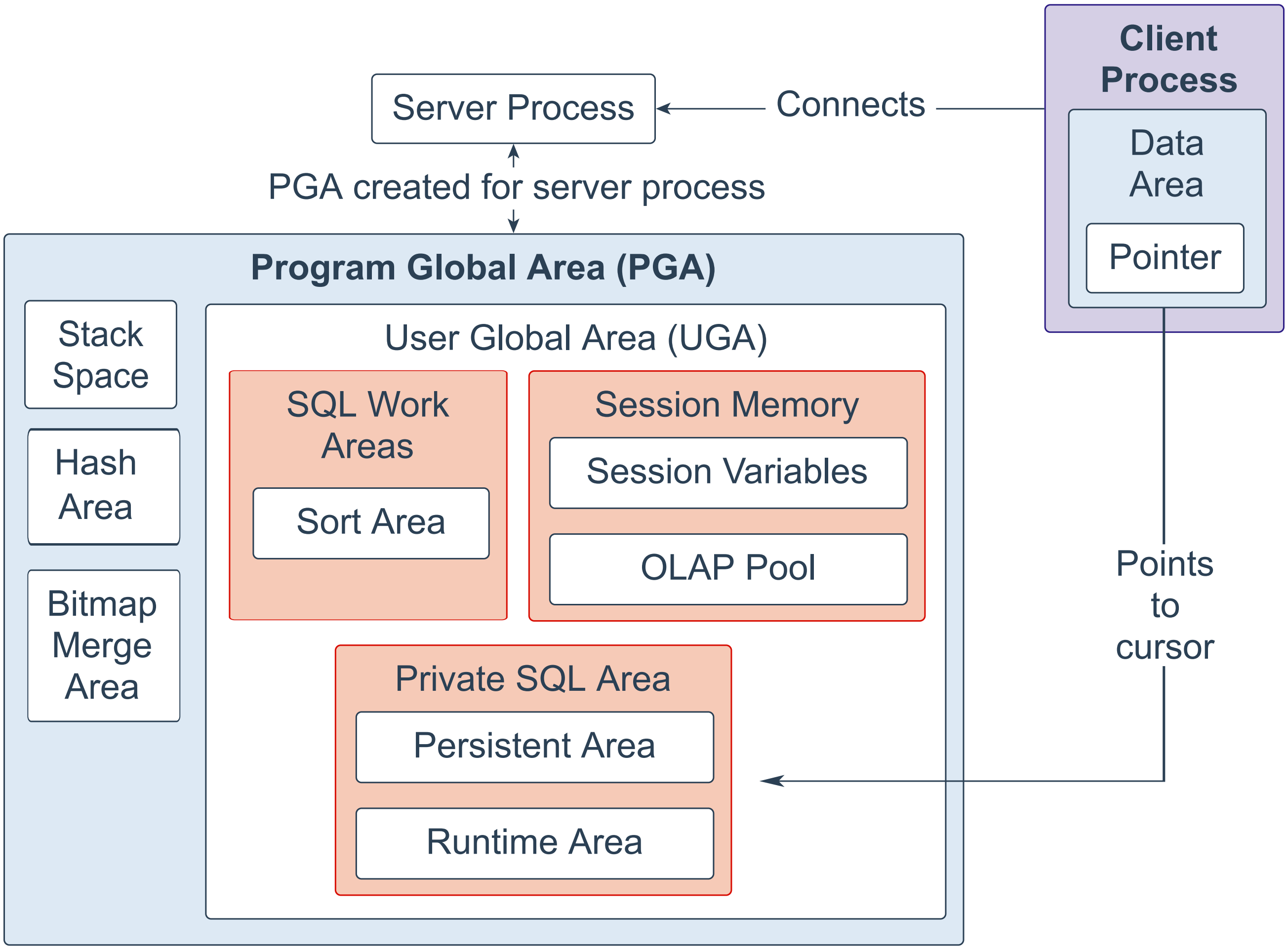
- **Shared pool:** Caches various constructs that can be shared among users; for example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information. The shared pool is involved in almost every operation that occurs in the database. For example, if a user executes a SQL statement, then Oracle Database accesses the shared pool.
- **Flashback buffer:** Is an optional component in the SGA. When Flashback Database is enabled, the background process called Recovery Writer Process (RVWR) is started. RVWR periodically copies modified blocks from the buffer cache to the flashback buffer, and sequentially writes Flashback Database data from the flashback buffer to the Flashback Database logs, which are circularly reused.
- **Database buffer cache:** Is the memory area that stores copies of data blocks read from data files. A buffer is a main memory address in which the buffer manager temporarily caches a currently or recently used data block. All users concurrently connected to a database instance share access to the buffer cache.
- **Database Smart Flash cache:** Is an optional memory extension of the database buffer cache for databases running on Solaris or Oracle Linux. It provides a level 2 cache for database blocks. It can improve response time and overall throughput for both read-intensive online transaction processing (OLTP) workloads and ad-hoc queries and bulk data modifications in a data warehouse (DW) environment. Database Smart Flash Cache resides on one or more flash disk devices, which are solid state storage devices that use flash memory. Database Smart Flash Cache is typically more economical than additional main memory, and is an order of magnitude faster than disk drives.
- **Redo log buffer:** Is a circular buffer in the SGA that holds information about changes made to the database. This information is stored in redo entries. Redo entries contain the information necessary to reconstruct (or redo) changes that are made to the database by data manipulation language (DML), data definition language (DDL), or internal operations. Redo entries are used for database recovery if necessary.
- **Large pool:** Is an optional memory area intended for memory allocations that are larger than is appropriate for the shared pool. The large pool can provide large memory allocations for the User Global Area (UGA)

for the shared server and the Oracle XA interface (used where transactions interact with multiple databases), message buffers used in the parallel execution of statements, buffers for Recovery Manager (RMAN) I/O workers, and deferred inserts.

- **In-Memory Area:** Is an optional component that enables objects (tables, partitions, and other types) to be stored in memory in a new format known as the columnar format. This format enables scans, joins, and aggregates to perform much faster than the traditional on-disk format, thus providing fast reporting and DML performance for both OLTP and DW environments. This feature is particularly useful for analytic applications that operate on a few columns returning many rows rather than for OLTP, which operates on a few rows returning many columns.
- **Memoptimize Pool:** Is an optional component that provides high performance and scalability for key-based queries. The Memoptimize Pool contains two parts, the memoptimize buffer area and the hash index. Fast lookup uses the hash index structure in the memoptimize pool providing fast access to the blocks of a given table (enabled for `MEMOPTIMIZE FOR READ)` permanently pinned in the buffer cache to avoid disk I/O. The buffers in the memoptimize pool are completely separate from the database buffer cache. The hash index is created when the Memoptimized Rowstore is configured, and is maintained automatically by Oracle Database.
- **Shared I/O pool (SecureFiles):** Is used for large I/O operations on SecureFile Large Objects (LOBs). LOBs are a set of data types that are designed to hold large amounts of data. SecureFile is an LOB storage parameter that allows deduplication, encryption, and compression.
- **Streams pool:** Is used by Oracle Streams, Data Pump, and GoldenGate integrated capture and apply processes. The Streams pool stores buffered queue messages, and it provides memory for Oracle Streams capture processes and apply processes. Unless you specifically configure it, the size of the Streams pool starts at zero. The pool size grows dynamically as needed when Oracle Streams is used.
- **Java pool:** Is used for all session-specific Java code and data in the Java Virtual Machine (JVM). Java pool memory is used in different ways, depending on the mode in which Oracle Database is running.
- **Fixed SGA:** Is an internal housekeeping area containing general information about the state of the database and database instance, and information communicated between processes.

For more information, see Overview of the System Global Area (SGA).

Program Global Area



The Program Global Area (PGA) is a non-shared memory region that contains data and control information exclusively for use by server and background processes. Oracle Database creates server processes to handle connections to the database on behalf of client programs. In a dedicated server environment, one PGA gets created for each server and background process that is started. Each PGA consists of stack space, hash area, bitmap merge area and a User Global Area (UGA). A PGA is deallocated when the associated server or background process using it is terminated.

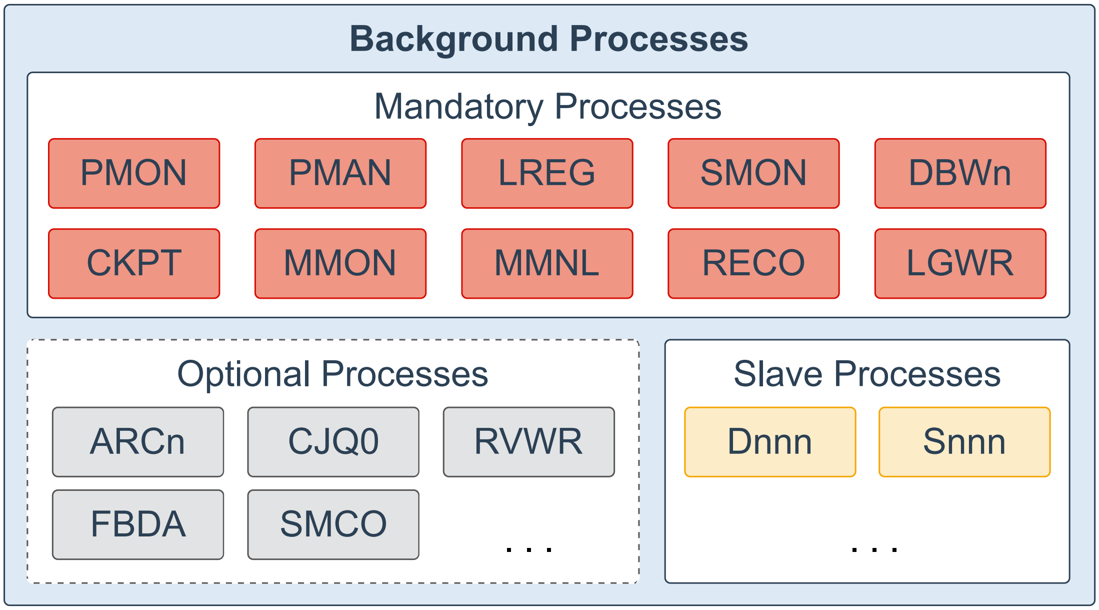
- In a shared server environment, multiple client users share the server process. The UGA is moved into the large pool, leaving the PGA with only stack space, hash area, and bitmap merge area.
- In a dedicated server session, the PGA consists of the following components:
  - SQL work areas: The sort area is used by functions that order data, such as `ORDER BY` and `GROUP BY`.
  - Session memory: This user session data storage area is allocated for session variables, such as logon information, and other information required by a database session. The OLAP pool manages OLAP data pages, which are equivalent to data blocks.
  - Private SQL area: This area holds information about a parsed SQL statement and other session-specific information for processing. When a server process executes SQL or PL/SQL code, the process

uses the private SQL area to store bind variable values, query execution state information, and query execution work areas. Multiple private SQL areas in the same or different sessions can point to a single execution plan in the SGA. The persistent area contains bind variable values. The run-time area contains query execution state information. A cursor is a name or handle to a specific area in the private SQL area. You can think of a cursor as a pointer on the client side and as a state on the server side. Because cursors are closely associated with private SQL areas, the terms are sometimes used interchangeably.

  - Stack space: Stack space is memory allocated to hold session variables and arrays.
  - Hash area: This area is used to perform hash joins of tables.
  - Bitmap merge area: This area is used to merge data retrieved from scans of multiple bitmap indexes.

For more information, see Overview of the Program Global Area (PGA).

Background Processes



Background processes are part of the database instance and perform maintenance tasks required to operate the database and to maximize performance for multiple users. Each background process performs a unique task, but works with the other processes. Oracle Database creates background processes automatically when you start a database instance. The background processes that are present depend on the features that are being used in the database. When you start a database instance, mandatory background processes automatically start. You can start optional background processes later as required.

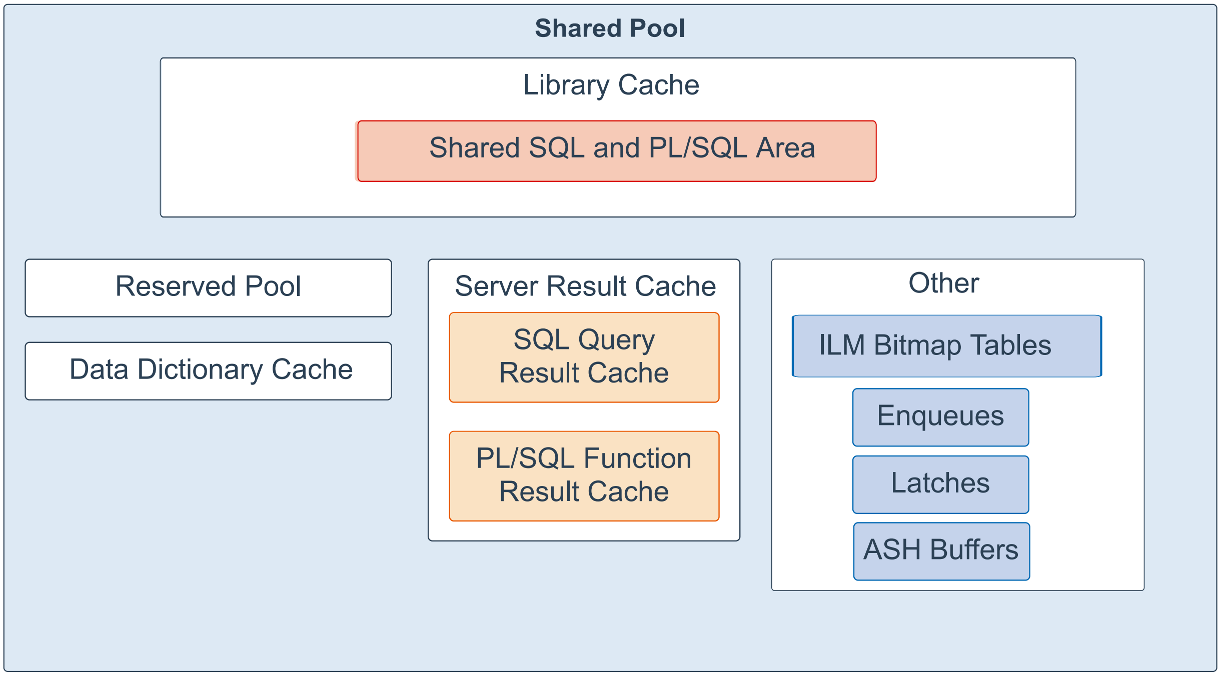
Mandatory background processes are present in all typical database configurations. These processes run by default in a read/write database instance started with a minimally configured initialization parameter file. A read-only database instance disables some of these processes. Mandatory background processes include the Process Monitor Process (PMON), Process Manager Process (PMAN), Listener Registration Process (LREG), System Monitor Process (SMON), Database Writer Process (DBWn), Checkpoint Process (CKPT), Manageability Monitor Process (MMON), Manageability Monitor Lite Process (MMNL), Recoverer Process (RECO), and Log Writer Process (LGWR).

Most optional background processes are specific to tasks or features. Some common optional processes include Archiver Processes (ARCn), Job Queue Coordinator Process (CJQ0), Recovery Writer Process (RVWR), Flashback Data Archive Process (FBDA), and Space Management Coordinator Process (SMCO).

Worker processes are background processes that perform work on behalf of other processes; for example, the Dispatcher Process (Dnnn) and Shared Server Process (Snnn).

For a complete list of background processes, see Background Processes.

Shared Pool



The shared pool is a component of the System Global Area (SGA) and is responsible for caching various types of program data. For example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information. The shared pool is involved in almost every operation that occurs in the database. For example, if a user executes a SQL statement, then Oracle Database accesses the shared pool.

The shared pool is divided into several subcomponents:

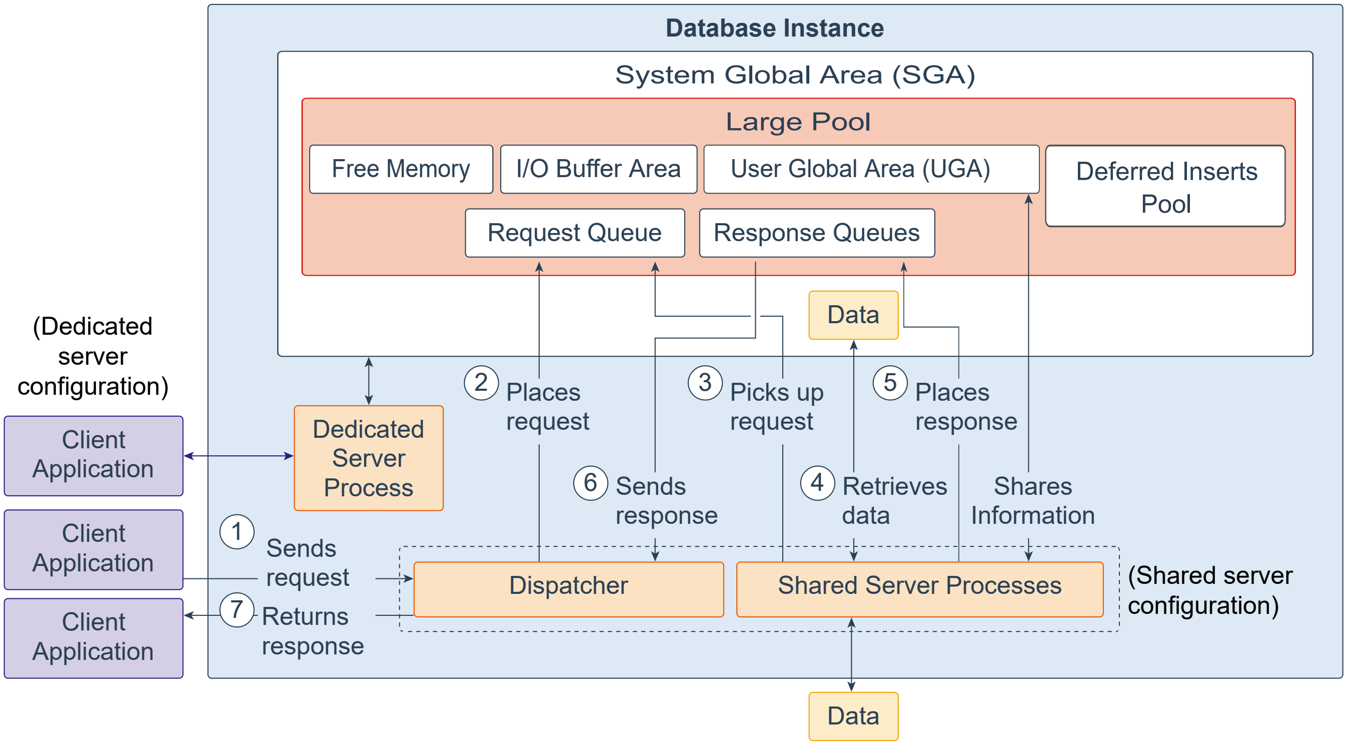
- **Library cache:** Is a shared pool memory structure that stores executable SQL and PL/SQL code. This cache contains the shared SQL and PL/SQL areas and control structures, such as locks and library cache handles. When a SQL statement is executed, the database attempts to reuse previously executed code. If a parsed representation of a SQL statement exists in the library cache and can be shared, the database reuses the code. This action is known as a soft parse or a library cache hit. Otherwise, the database must build a new executable version of the application code, which is known as a hard parse or a library cache miss.
- **Reserved pool:** Is a memory area in the shared pool that Oracle Database can use to allocate large contiguous chunks of memory. The database allocates memory from the shared pool in chunks. Chunking allows large objects (over 5 KB) to be loaded into the cache without requiring a single contiguous area. In this way, the database reduces the possibility of running out of contiguous memory because of fragmentation.
- **Data dictionary cache:** Stores information about database objects (that is, dictionary data). This cache is also known as the row cache because it holds data as rows instead of buffers.
- **Server result cache:** Is a memory pool within the shared pool and holds result sets. The server result cache contains the SQL query result cache and PL/SQL function result cache, which share the same infrastructure. The SQL query result cache stores the results of queries and query fragments. Most applications benefit

from this performance improvement. The PL/SQL function result cache stores function result sets. Good candidates for result caching are frequently invoked functions that depend on relatively static data.

- **Other components:** Include enqueues, latches, Information Lifecycle Management (ILM) bitmap tables, Active Session History (ASH) buffers, and other minor memory structures. Enqueues are shared memory structures (locks) that serialize access to database resources. They can be associated with a session or transaction. Examples are: Controlfile Transaction, Datafile, Instance Recovery, Media Recovery, Transaction Recovery, Job Queue, and so on. Latches are used as a low-level serialization control mechanism used to protect shared data structures in the SGA from simultaneous access. Examples are row cache objects, library cache pin, and log file parallel write.

For more information, see Shared Pool.

Large Pool



The large pool is an optional memory area that database administrator's can configure to provide large memory allocations for the following:

- **User Global Area (UGA):** Session memory for the shared server and the Oracle XA interface (used where transactions interact with multiple databases)
- **I/O Buffer Area:** I/O server processes, message buffers used in parallel query operations, buffers for Recovery Manager (RMAN) I/O workers, and advanced queuing memory table storage
- **Deferred Inserts Pool:** The fast ingest feature enables high-frequency, single-row data inserts into database for tables defined as `MEMOPTIMIZE FOR WRITE`. The inserts by fast ingest are also known as deferred inserts. They are initially buffered in the large pool and later written to disk asynchronously by the Space Management Coordinator (SMCO) and Wxxx worker background processes after 1MB worth of writes per session per object or after 60 seconds. Any data buffered in this pool, even committed, cannot be read by any session, including the writer, until the SMCO background process sweeps.
  The pool is initialized in the large pool at the first inserted row of a memoptimized table. 2G is allocated from the large pool when there is enough space. If there is not enough space in the large pool, an `ORA-4031` is internally discovered and automatically cleared. The allocation is retried with half the requested size. If there is still not enough space in the large pool, the allocation is retried with 512M and 256M after which the feature is disabled until the instance is restarted. Once the pool is initialized, the size remains static. It cannot grow or shrink.
- **Free memory**

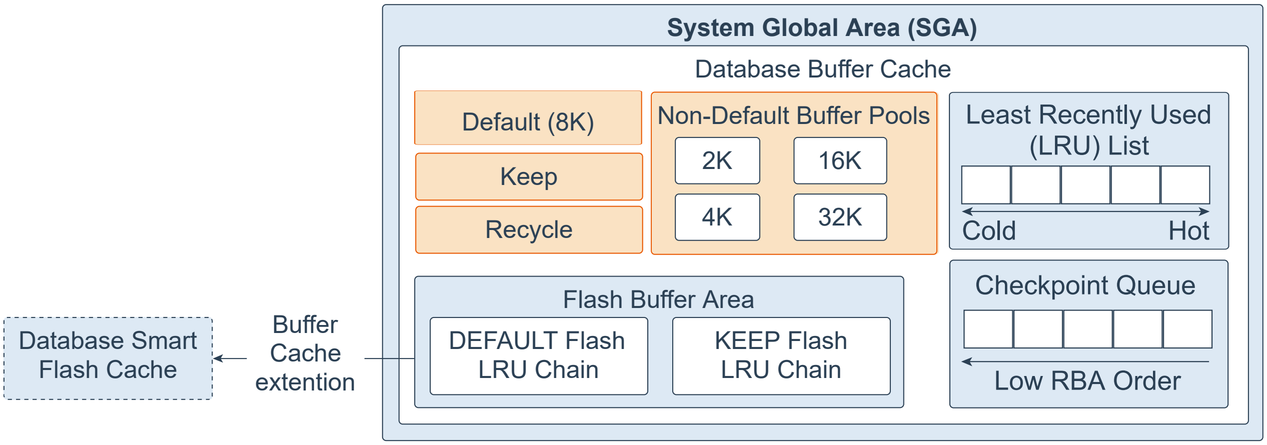
The large pool is different from reserved space in the shared pool, which uses the same Least Recently Used (LRU) list as other memory allocated from the shared pool. The large pool does not have an LRU list. Pieces of memory are allocated and cannot be freed until they are done being used.

A request from a user is a single API call that is part of the user's SQL statement. In a dedicated server environment, one server process handles requests for a single client process. Each server process uses system resources, including CPU cycles and memory. In a shared server environment, the following actions occur:

1. A client application sends a request to the database instance, and that request is received by the dispatcher.
2. The dispatcher places the request on the request queue in the large pool.
3. The request is picked up by the next available shared server process. The shared server processes check the common request queue for new requests, picking up new requests on a first-in-first-out basis. One shared server process picks up one request in the queue.
4. The shared server process makes all the necessary calls to the database to complete the request. First, the shared server process accesses the library cache in the shared pool to verify the requested items; for example, it checks whether the table exists, whether the user has the correct privileges, and so on. Next, the shared server process accesses the buffer cache to retrieve the data. If the data is not there, the shared server process accesses the disk. A different shared server process can handle each database call. Therefore, requests to parse a query, fetch the first row, fetch the next row, and close the result set may each be processed by a different shared server process. Because a different shared server process may handle each database call, the User Global Area (UGA) must be a Shared Memory area, as the UGA contains information about each client session. Or reversed, the UGA contains information about each client session and must be available to all shared server processes because any shared server process may handle any session's database call.
5. After the request is completed, a shared server process places the response on the calling dispatcher's response queue in the large pool. Each dispatcher has its own response queue.
6. The response queue sends the response to the dispatcher.
7. The dispatcher returns the completed request to the appropriate client application.

For more information, see Large Pool.

Database Buffer Cache



The database buffer cache, also called the buffer cache, is the memory area in the System Global Area (SGA) that stores copies of data blocks read from data files. A buffer is a database block-sized chunk of memory. Each buffer has an address called a Database Buffer Address (DBA). All users concurrently connected to a database instance share access to the buffer cache. The goals of the buffer cache is to optimize physical I/O and to keep frequently accessed blocks in the buffer cache and write infrequently accessed blocks to disk.

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

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

The database buffer cache consists of the following:

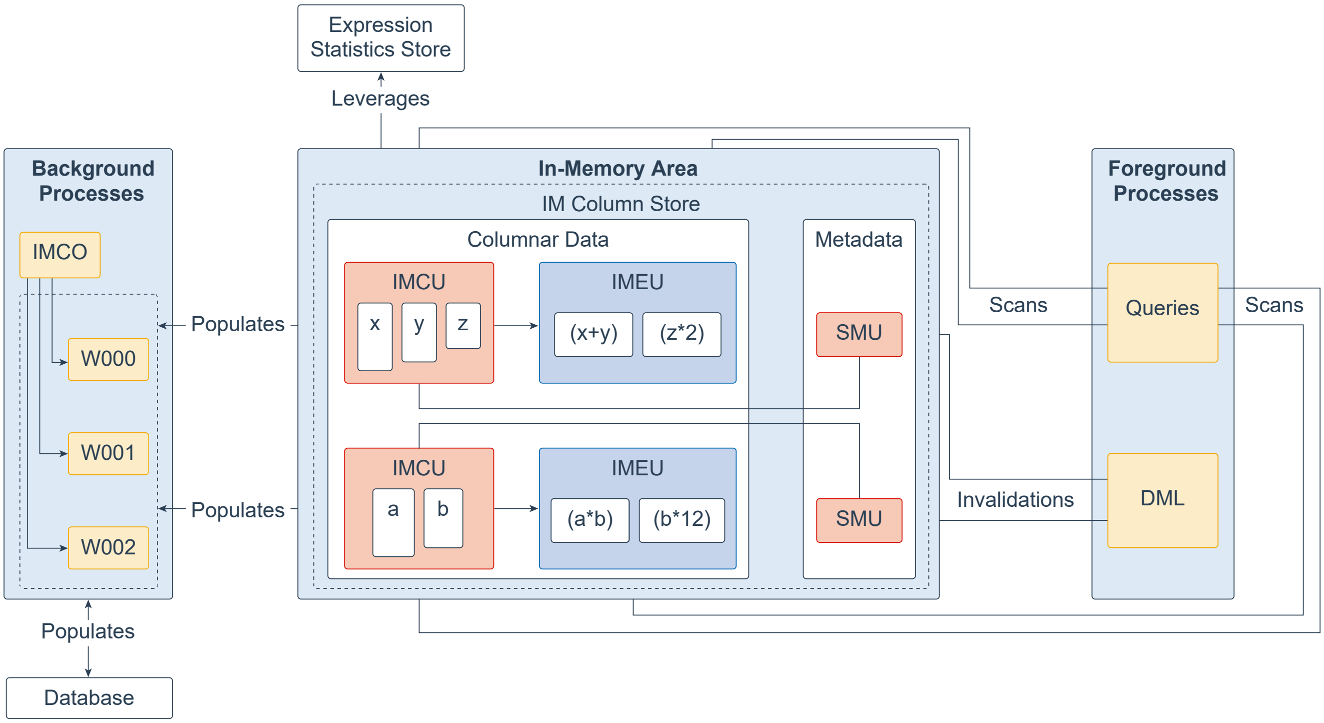
- **Default pool:** Is the location where blocks are normally cached. The default block size is 8 KB. Unless you manually configure separate pools, the default pool is the only buffer pool. The optional configuration of the other pools has no effect on the default pool.
- **Keep pool:** Is intended for blocks that were accessed frequently, but which aged out of the default pool because of lack of space. The purpose of the keep buffer pool is to retain specified objects in memory, thus avoiding I/O operations.
- **Recycle pool:** Is intended for blocks that are used infrequently. A recycle pool prevents specified objects from consuming unnecessary space in the cache.
- **Non-default buffer pools:** Are for tablespaces that use the nonstandard block sizes of 2 KB, 4 KB, 16 KB, and 32 KB. Each non-default block size has its own pool. Oracle Database manages the blocks in these pools in the same way as in the default pool.
- **Database Smart Flash Cache (flash cache):** Lets you use flash devices to increase the effective size of the buffer cache without adding more main memory. Flash cache can improve database performance by having

the database cache's frequently accessed data stored into flash memory instead of reading the data from magnetic disk. When the database requests data, the system first looks in the database buffer cache. If the data is not found, the system then looks in the Database Smart Flash Cache buffer. If it does not find the data there, only then does it look in disk storage. You must configure a flash cache on either all or none of the instances in an Oracle Real Application Clusters environment.

- **Least Recently Used list (LRU):** Contains pointers to dirty and non-dirty buffers. The LRU list has a hot end and cold end. A cold buffer is a buffer that has not been recently used. A hot buffer is frequently accessed and has been recently used. Conceptually, there is only one LRU, but for data concurrency the database actually uses several LRUs.
- **Checkpoint queue**
- **Flash Buffer Area:** Consists of a DEFAULT Flash LRU Chain and a KEEP Flash LRU Chain. Without Database Smart Flash Cache, when a process tries to access a block and the block does not exist in the buffer cache, the block will first be read from disk into memory (physical read). When the in-memory buffer cache gets full, a buffer will get evicted out of the memory based on a least recently used (LRU) mechanism. With Database Smart Flash Cache, when a clean in-memory buffer is aged out, the buffer content is written to the flash cache in the background by the Database Writer process (DBWn), and the buffer header is kept in memory as metadata in either the DEFAULT flash or KEEP Flash LRU list, depending on the value of the `FLASH_CACHE` object attribute. The KEEP flash LRU list is used to maintain the buffer headers on a separate list to prevent the regular buffer headers from replacing them. Thus, the flash buffer headers belonging to an object specified as `KEEP` tend to stay in the flash cache longer. If the `FLASH_CACHE` object attribute is set to `NONE`, the system does not retain the corresponding buffers in the flash cache or in memory. When a buffer that was already aged out of memory is accessed again, the system checks the flash cache. If the buffer is found, it reads it back from the flash cache which takes only a fraction of the time of reading from the disk. The consistency of flash cache buffers across Real Application Clusters (RAC) is maintained in the same way as by Cache Fusion. Because the flash cache is an extended cache and direct path I/O totally bypasses the buffer cache, this feature does not support direct path I/O. Note that the system does not put dirty buffers in flash cache because it may have to read buffers into memory in order to checkpoint them because writing to flash cache does not count for checkpoint.

For more information, see Database Buffer Cache.

In-Memory Area



The In-Memory Area is an optional SGA component that contains the In-Memory column store (IM column store), which stores tables and partitions in memory using a columnar format optimized for rapid scans. The IM column store enables data to be simultaneously populated in the SGA in both the traditional row format (in the buffer cache) and a columnar format. The database transparently sends online transactional processing (OLTP) queries, such as primary key lookups, to the buffer cache, and analytic and reporting queries to the IM column store. When fetching data, Oracle Database can also read data from both memory areas within the same query. The dual-format architecture does not double memory requirements. The buffer cache is optimized to run with a much smaller size than the size of the database.

You should populate only the most performance-critical data in the IM column store. To add an object to the IM column store, turn on the `INMEMORY` attribute for an object when creating or altering it. You can specify this attribute on a tablespace (for all new tables and views in the tablespace), table, (sub)partition, materialized view, or subset of columns within an object.

The IM column store manages both data and metadata in optimized storage units, not in traditional Oracle data blocks. An In-Memory Compression Unit (IMCU) is a compressed, read-only storage unit that contains data for one or more columns. A Snapshot Metadata Unit (SMU) contains metadata and transactional information for an associated IMCU. Every IMCU maps to a separate SMU.

The Expression Statistics Store (ESS) is a repository that stores statistics about expression evaluation. The ESS resides in the SGA and also persists on disk. When an IM column store is enabled, the database leverages the ESS for its In-Memory Expressions (IM expressions) feature. An In-Memory Expression Unit (IMEU) is a storage container for materialized IM expressions and user-defined virtual columns. Note that the ESS is independent of the IM column store. The ESS is a permanent component of the database and cannot be disabled.

Conceptually, an IMEU is a logical extension of its parent IMCU. Just as an IMCU can contain multiple columns, an IMEU can contain multiple virtual columns. Every IMEU maps to exactly one IMCU, mapping to the same row set. The IMEU contains expression results for the data contained in its associated IMCU. When the IMCU is populated, the associated IMEU is also populated.

A typical IM expression involves one or more columns, possibly with constants, and has a one-to-one mapping with the rows in the table. For example, an IMCU for an `EMPLOYEES` table contains rows 1-1000 for the column `weekly_salary`. For the rows stored in this IMCU, the IMEU calculates the automatically detected IM expression `weekly_salary*52`, and the user-defined virtual column `quarterly_salary` defined as `weekly_salary*12`. The third row down in the IMCU maps to the third row down in the IMEU.

The In-Memory area is sub-divided into two pools: a 1MB columnar data pool used to store the actual column-formatted data populated into memory (IMCUs and IMEUs), and a 64K metadata pool used to store metadata about the objects that are populated into the IM column store. The relative size of the two pools is determined by internal heuristics; the majority of the In-Memory area memory is allocated to the 1MB pool. The size of the In-Memory area is controlled by the initialization parameter `INMEMORY_SIZE` (default 0) and must have a minimum size of 100MB. Starting in Oracle Database 12.2, you can increase the size of the In-Memory area on the fly by increasing the `INMEMORY_SIZE` parameter via an `ALTER SYSTEM` command by at least 128MB. Note that it is not possible to shrink the size of the In-Memory area on the fly.

Database In-Memory has a new "Base Level" feature that allows its use with up to a 16GB column store without triggering any license tracking.

An in-memory table gets IMCUs allocated in the IM column store at first table data access or at database startup. An in-memory copy of the table is made by doing a conversion from the on-disk format to the new in-memory columnar format. This conversion is done each time the instance restarts as the IM column store copy resides only in memory. When this conversion is done, the in-memory version of the table gradually becomes available for queries. If a table is partially converted, queries are able to use the partial in-memory version and go to disk for the rest, rather than waiting for the entire table to be converted.

In-memory hybrid scans can access some data from the IM column store, and some data from the row store, when not all columns in a table have been populated into the In-Memory Column Store, improving performance by orders of magnitude over pure row store queries.

Automatic In-Memory enables, populates, evicts, and recompresses segments without user intervention. When `INMEMORY_AUTOMATIC_LEVEL` is set to `HIGH`, the database automatically enables and populates segments based on their usage patterns. This automation helps maximize the number of objects that can be populated into the In-Memory Column Store at one time.

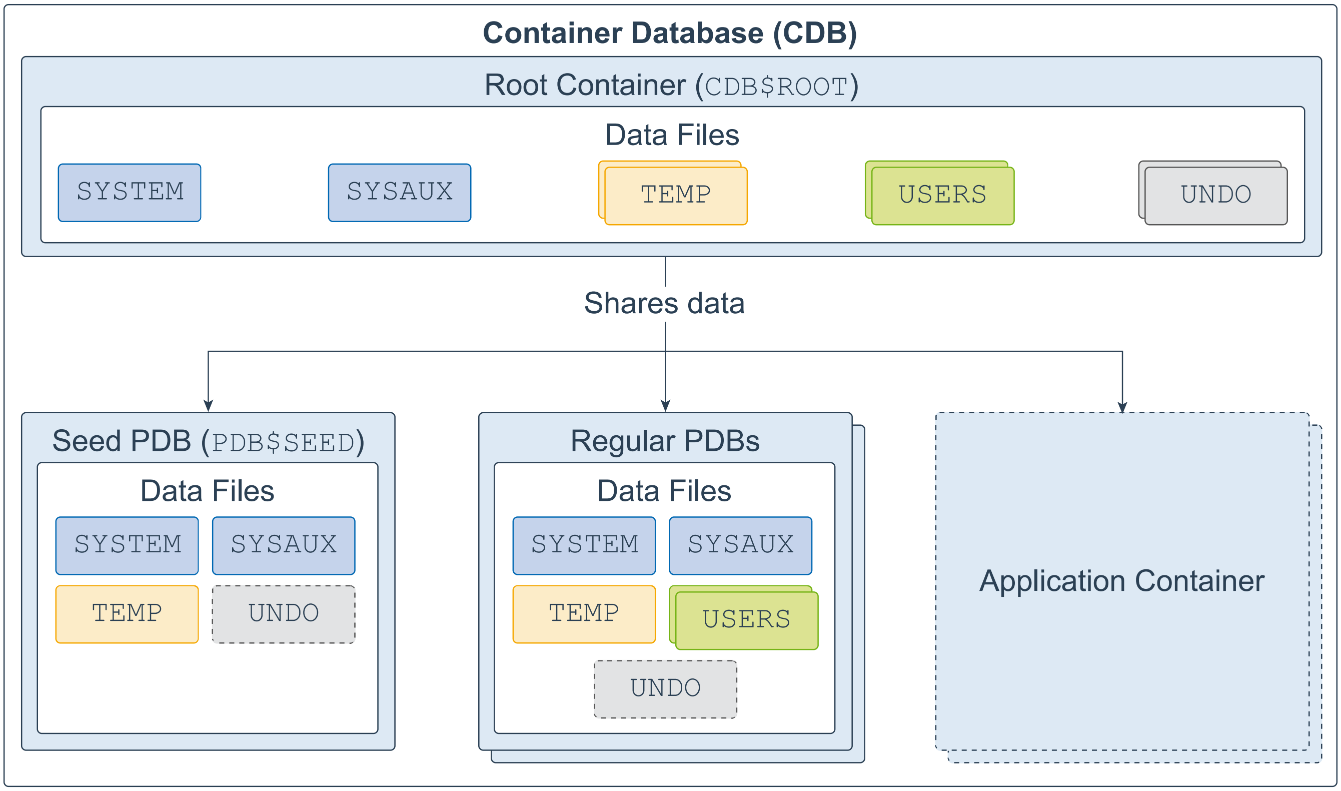
In response to queries and data manipulation language (DML), server processes scan columnar data and update SMU metadata. Background processes populate row data from disk into the IM column store. The In-Memory Coordinator Process (IMCO) is a background process that initiates background population and repopulation of columnar data. The Space Management Coordinator Process (SMCO) and Space Management Worker Processes (Wnnn) are background processes that do the actual populating and repopulating of data on behalf of IMCO. DML block changes are written to the buffer cache, and then to disk. Background processes then repopulate row data from disk into the IM column store based on the metadata invalidations and query requests.

You can enable the In-Memory FastStart feature to write the columnar data in the IM Column Store back to a tablespace in the database in compressed columnar format. This feature makes database startup faster. Note that this feature does not apply to IMEUs. They are always populated dynamically from the IMCUs.

In-Memory deep vectorization can optimize complex SQL operators by pipelining the physical operators inside each SQL operator and vectorizing them using SIMD techniques. This feature is enabled by default but can be disabled by setting the `INMEMORY_DEEP_VECTORIZATION` initialization parameter to false.

For more information, see Introduction to Oracle Database In-Memory.

Database Data Files



A multitenant container database (CDB) is a set of physical files that store user data and metadata. The metadata consists of structural, configuration, and control information about the database server.

A CDB is made up of one CDB root container (also called the root), exactly one seed pluggable database (seed PDB), zero or more user-created pluggable databases (simply referred to as PDBs), and zero or more application containers. The entire CDB is referred to as the system container. To a user or application, PDBs appear logically as separate databases.

The CDB root, named CDB$ROOT, contains multiple data files, control files, redo log files, flashback logs, and archived redo log files. The data files store Oracle-supplied metadata and common users (users that are known in every container), which are shared with all PDBs.

The seed PDB, named PDB$SEED, is a system-supplied PDB template containing multiple data files that you can use to create new PDBs.

The regular PDB contains multiple data files that contain the data and code required to support an application; for example, a Human Resources application. Users interact only with the PDBs, and not the seed PDB or root container. You can create multiple PDBs in a CDB. One of the goals of the multitenant architecture is that each PDB has a one-to-one relationship with an application.

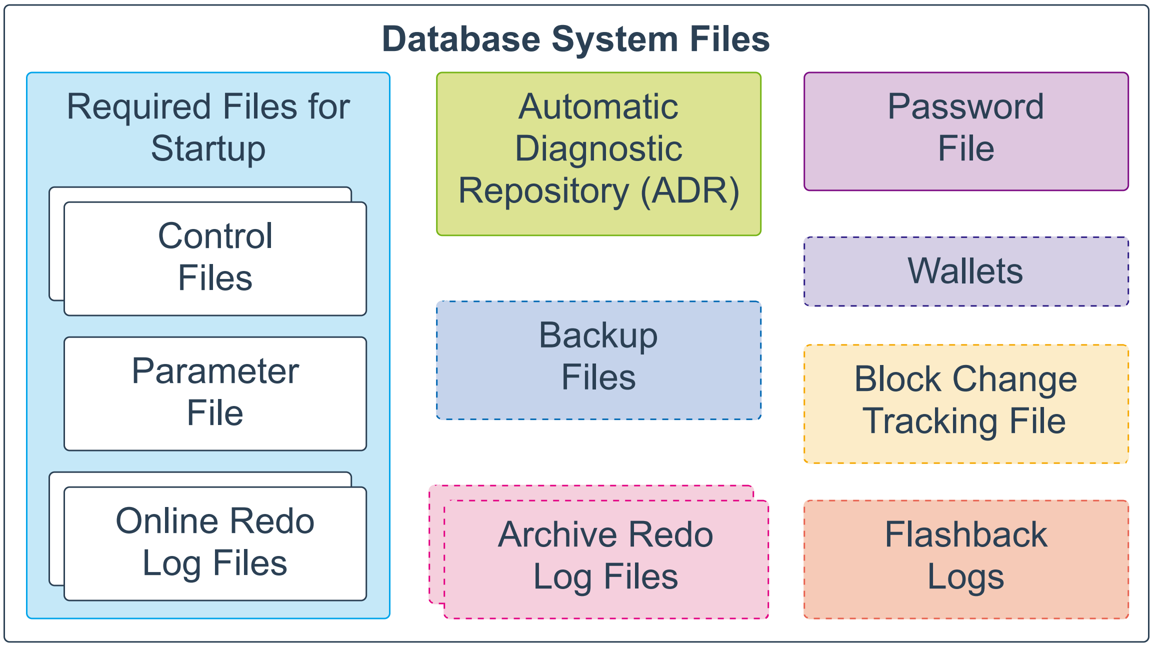
An application container is an optional collection of PDBs within a CDB that stores data for an application. The purpose of creating an application container is to have a single master application definition. You can have multiple application containers in a CDB.

A database is divided into logical storage units called tablespaces, which collectively store all the database data. Each tablespace represents one or more data files. The root container and regular PDBs have a SYSTEM, SYSAUX, USERS, TEMP, and UNDO tablespace (optional in a regular PDB). A seed PDB has a SYSTEM, SYSAUX, TEMP, and optional UNDO tablespace.

Non-CDBs are desupported in Oracle Database 21c which means that the Oracle Universal Installer and DBCA can no longer create non-CDB Oracle Database instances.

For more information, see Introduction to the Multitenant Architecture.

Database System Files



The following database system files are used during the operation of an Oracle Database and reside on the database server. Note that data files are physical files that belong to database containers and are not described here.

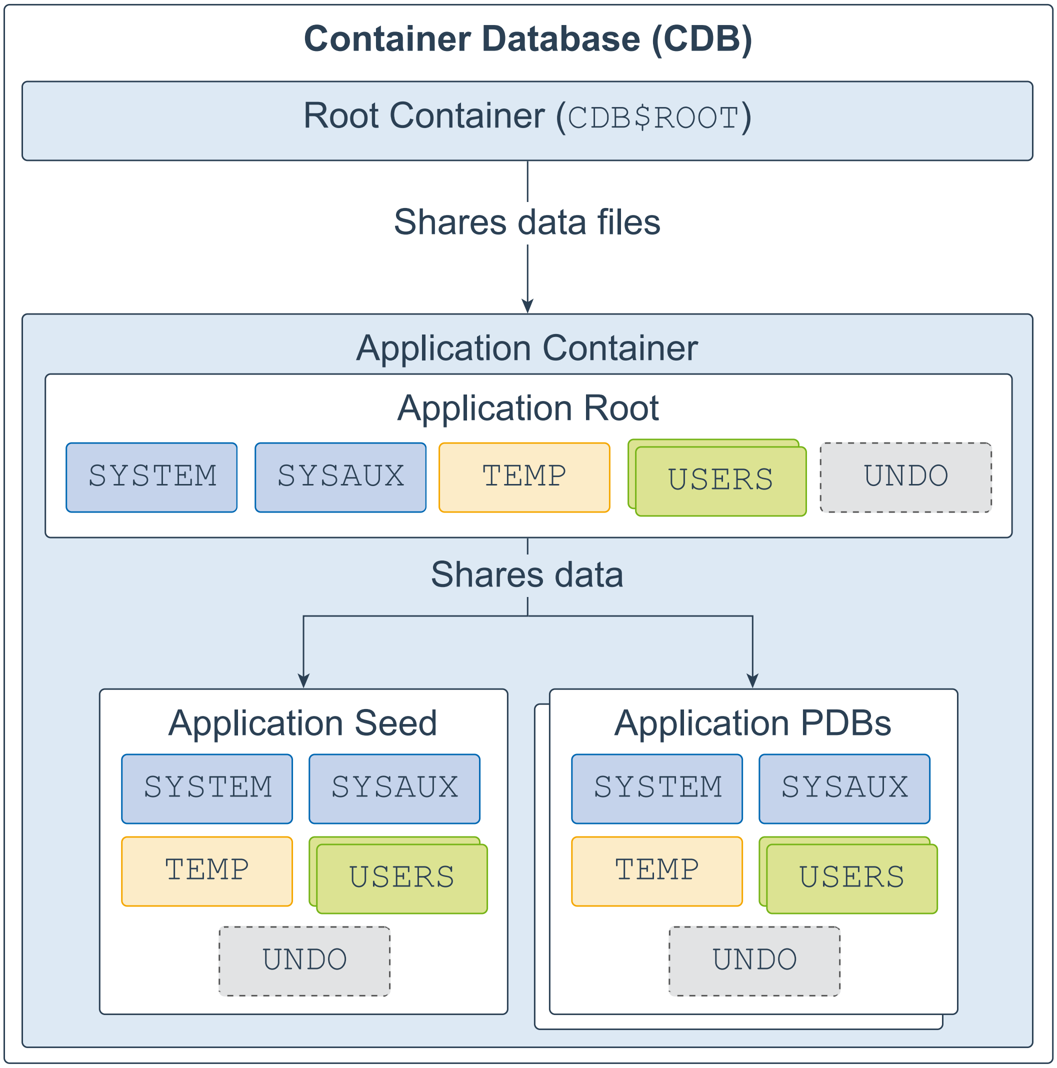
- **Control files:** A control file is a required file that stores metadata about the data files and online redo log files; for example, their names and statuses. This information is required by the database instance to open the database. Control files also contain metadata that must be accessible when the database is not open. It is highly recommended that you make several copies of the control file in your database server for high availability.
- **Parameter file:** This required file defines how the database instance is configured when it starts up. It can be either an initialization parameter file (pfile) or a server parameter file (spfile).
- **Online redo log files:** These required files store changes to the database as they occur and are used for data recovery.
- **Automatic Diagnostic Repository (ADR):** The ADR is a file-based repository for database diagnostic data, such as traces, dumps, the alert log, health monitor reports, and more. It has a unified directory structure across multiple instances and multiple products. The database, Oracle Automatic Storage Management (Oracle ASM), the listener, Oracle Clusterware, and other Oracle products or components store all diagnostic data in the ADR. Each instance of each product stores diagnostic data underneath its own home directory within the ADR.
- **Backup files:** These optional files are used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.
- **Archived redo log files:** These optional files contain an ongoing history of the data changes that are generated by the database instance. Using these files and a backup of the database, you can recover a lost data file. That is, archive logs enable the recovery of restored data files.

- **Password file:** This optional file enables users using the SYSDBA, SYSOPER, SYSBACKUP, SYSDG, SYSKM, SYSRAC, and SYSASM roles to connect remotely to the database instance and perform administrative tasks.
- **Wallets:** For large-scale deployments where applications use password credentials to connect to databases, it is possible to store such credentials in a client-side Oracle wallet. An Oracle wallet is a secure software container that is used to store authentication and signing credentials. Possible wallets include an Oracle wallet for user credentials, Encryption Wallet for Transparent Data Encryption (TDE), and an Oracle Public Cloud (OPC) wallet for the database backup cloud module. A wallet is optional, but recommended.
- **Block change tracking file:** Block change tracking improves the performance of incremental backups by recording changed blocks in the block change tracking file. During an incremental backup, instead of scanning all data blocks to identify which blocks have changed, Oracle Recovery Manager (RMAN) uses this file to identify the changed blocks that need to be backed up. A block change tracking file is optional.
- **Flashback logs:** Flashback Database is similar to conventional point-in-time recovery in its effects. It enables you to return a database to its state at a time in the recent past. Flashback Database uses its own logging mechanism, creating flashback logs and storing them in the fast recovery area. You can use Flashback Database only if flashback logs are available. To take advantage of this feature, you must set up your database in advance to create flashback logs. Flashback logs are optional.

Control files, online redo log files, and archive redo log files can be multiplexed, which mean that two or more identical copies can be automatically maintained in separate locations.

The control file, parameter file, and online redo log files are required for database startup. For more information, see Physical Storage Structures.

Application Containers



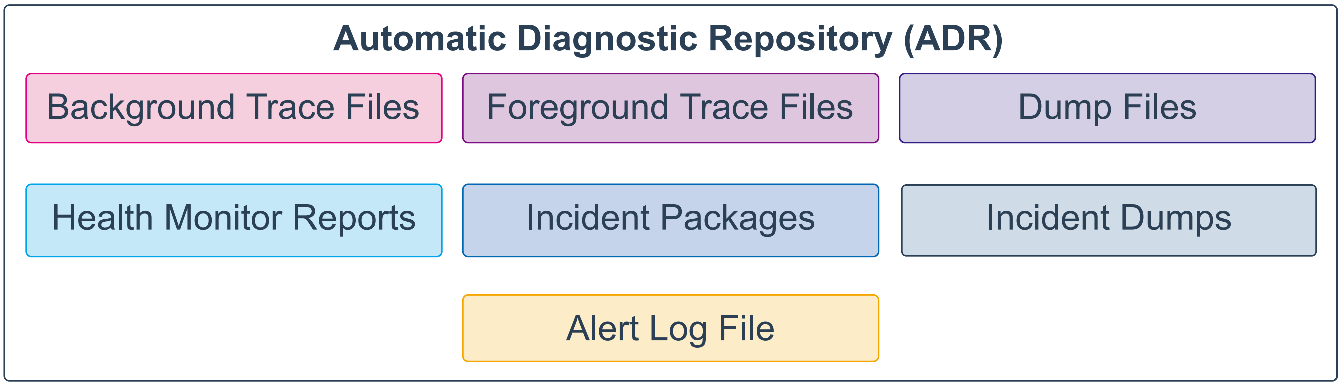
An application container is an optional, user-created CDB component that stores data and metadata for application PDBs. A CDB can include zero or more application containers. An application container consists of exactly one application root and one or more application PDBs, which plug into the CDB root. An application root belongs to the CDB root and no other container, and stores the common metadata and data.

A typical application installs application common users, metadata-linked common objects, and data-linked common objects. You might create multiple sales-related PDBs within one application container, with these PDBs sharing an application back end that consists of a set of common tables and table definitions.

The application root, application seed, and application PDB each have a `SYSTEM`, `SYSAUX`, `TEMP`, `USERS`, and optional `UNDO` tablespace. Each tablespace represents one or more data files.

For more information, see About Application Containers.

Automatic Diagnostic Repository (ADR)



Automatic Diagnostic Repository (ADR)

Background Trace Files

Foreground Trace Files

Dump Files

Health Monitor Reports

Incident Packages

Incident Dumps

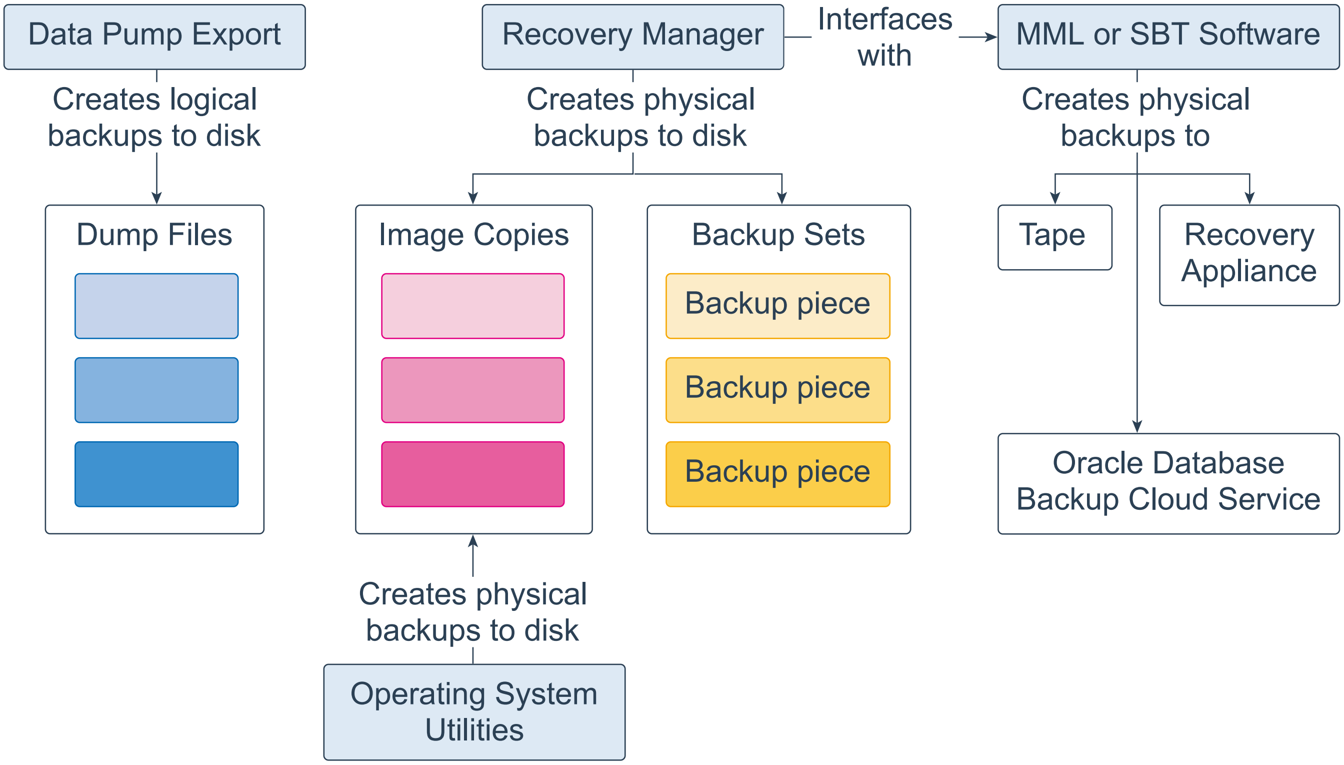
Alert Log File

The Automatic Diagnostic Repository (ADR) is a system-wide tracing and logging central repository for database diagnostic data. It includes the following items:

- **Background trace files:** Each database background process can write to an associated trace file. When a process detects an internal error, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, whereas other information is for Oracle Support Services. Typically, database background process trace file names contain the Oracle system identifier (SID), the background process name, and the operating system process number. An example of a trace file for the RECO process is `mytest_reco_10355.trc`.
- **Foreground trace files:** Each server process can write to an associated trace file. When a process detects an internal error, the process dumps information about the error to its trace file. Server process trace file names contain the Oracle SID, the string `ora`, and the operating system process number. An example of a server process trace file name is `mytest_ora_10304.trc`.
- **Dump files:** A diagnostic dump file is a special type of trace file that contains detailed point-in-time information about a state or structure. A dump file is typically a one-time output of diagnostic data in response to an event whereas a trace file tends to be a continuous output of diagnostic data.
- **Health monitor reports:** Oracle Database includes a framework called Health Monitor for running diagnostic checks on the database. Health checks detect file corruptions, physical and logical block corruptions, undo and redo corruptions, data dictionary corruptions, and more. The health checks generate reports of their findings and, in many cases, recommendations for resolving problems.
- **Incident packages:** For the customized approach to uploading diagnostic data to Oracle Support, you first collect the data into an intermediate logical structure called an incident package (package). A package is a collection of metadata that is stored in the ADR and points to diagnostic data files and other files both in and outside of the ADR. When you create a package, you select one or more problems to add to the package. The Support Workbench then automatically adds to the package the problem information, incident information, and diagnostic data (such as trace files and dumps) associated with the selected problems.
- **Incident dumps:** When an incident occurs, the database writes one or more dumps to the incident directory created for the incident. Incident dumps also contain the incident number in the file name.
- **Alert log file:** The alert log of a database is an chronological log of messages and errors. Oracle recommends that you review the alert log periodically.

For more information, see Automatic Diagnostic Repository.

Backup Files


Data Pump Export
Creates logical backups to disk
Dump Files
Recovery Manager
Interfaces with
MML or SBT Software
Creates physical backups to disk
Image Copies
Backup Sets
Backup piece
Backup piece
Backup piece
Creates physical backups to
Tape
Recovery Appliance
Oracle Database Backup Cloud Service
Creates physical backups to disk
Operating System Utilities

Database backups can be either physical or logical.

- **Physical backups** are copies of physical database files. You can make physical backups with Recovery Manager (RMAN) or operating system utilities.
- **Logical backups** contain tables, stored procedures, and other logical data. You can extract logical data with an Oracle Database utility, such as Data Pump Export, and store it in a binary file. Logical backups can supplement physical backups.

Database backups created by RMAN are stored as image copies or backup sets.

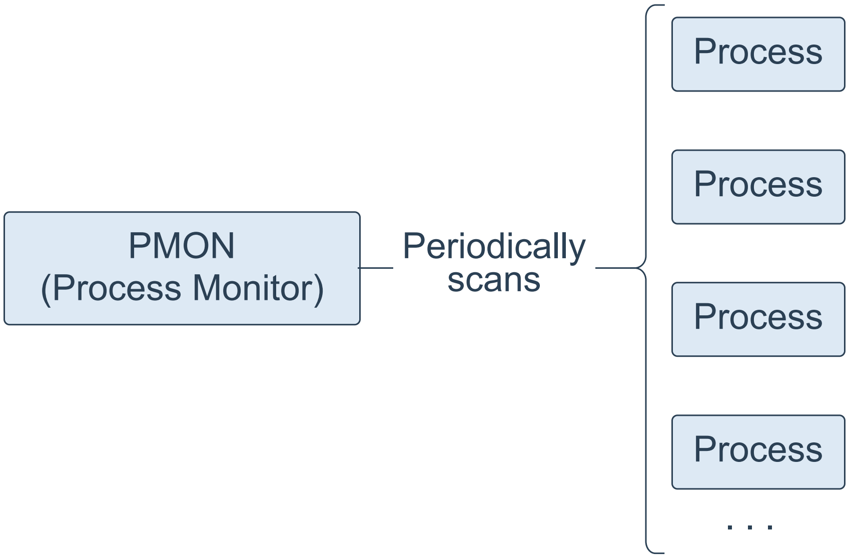
- An **image copy** is a bit-for-bit, on-disk duplicate of a data file, control file, or archived redo log file. You can create image copies of physical files with operating system utilities or RMAN and use either tool to restore them. Image copies are useful for disk because you can update them incrementally and recover them in place.
- A **backup set** is a proprietary format created by RMAN that contains the data from one or more data files, archived redo log files, control files, or server parameter file. The smallest unit of a backup set is a binary file called a backup piece. Backup sets are the only form in which RMAN can write backups to sequential devices, such as tape drives. One advantage of backup sets is that RMAN uses unused block compression to save space in backing up data files. Only those blocks in the data files that have been used to store data are included in the backup set. Backup sets can also be compressed, encrypted, sent to tape, and use advanced unused-space compression that is not available with datafile copies.

RMAN can interface with Media Management Library (MML) or System Backup to Tape (SBT) software, which can create backups to tape, Oracle Database Backup Cloud Service, or Zero Data Loss Recovery Appliance (commonly known as Recovery Appliance).

For more information, see:

- Backup and Recovery
- About Zero Data Loss Recovery Appliance

Process Monitor Process (PMON)

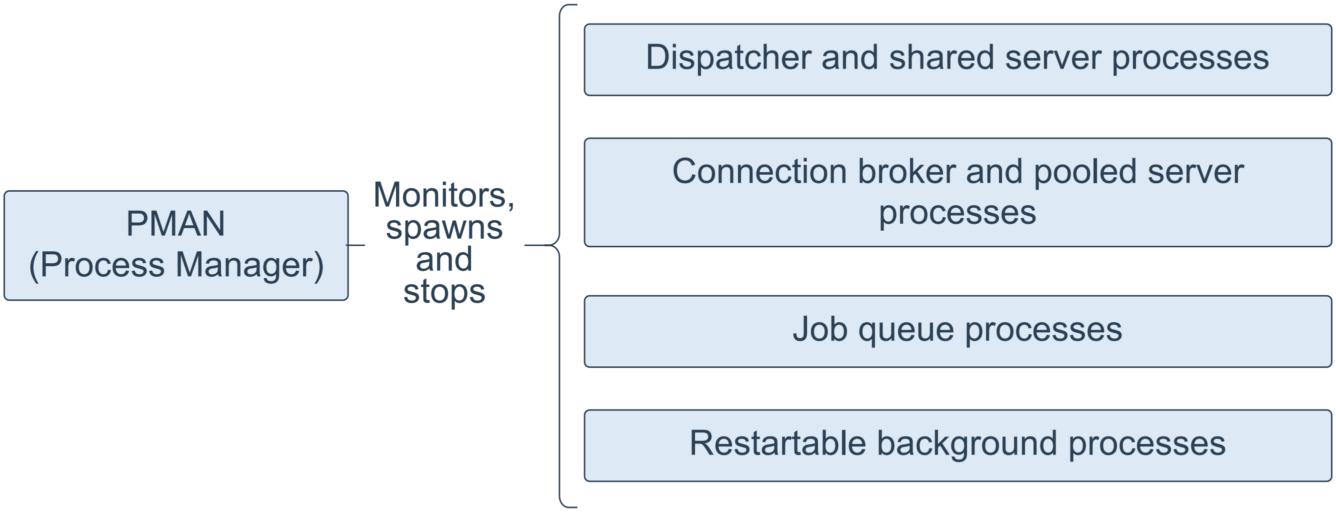


Process Monitor Process (PMON) is a background process that periodically scans all processes to find any that have died abnormally. PMON is then responsible for coordinating cleanup performed by the Cleanup Main Process (CLMN) and the Cleanup Worker Process workers (CLnn).

PMON runs as an operating system process, and not as a thread. In addition to database instances, PMON also runs on Oracle Automatic Storage Management (ASM) instances and Oracle ASM Proxy instances.

For a complete list of background processes, see Background Processes.

Process Manager Process (PMAN)


PMAN
(Process Manager)
Monitors, spawns and stops
Dispatcher and shared server processes
Connection broker and pooled server processes
Job queue processes
Restartable background processes

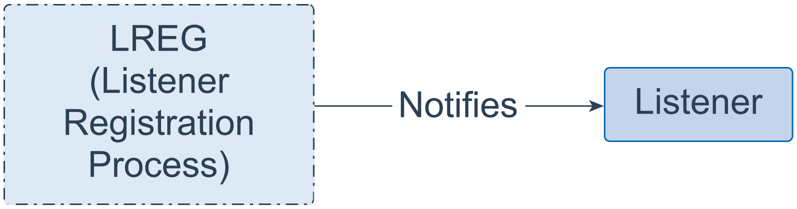
Process Manager Process (PMAN) is a background process that monitors, spawns, and stops the following as needed:

- Dispatcher and shared server processes
- Connection broker and pooled server processes for database resident connection pools
- Job queue processes
- Restartable background processes

PMAN runs as an operating system process, and not as a thread. In addition to database instances, PMAN also runs on Oracle Automatic Storage Management (ASM) instances and Oracle ASM Proxy instances.

For a complete list of background processes, see Background Processes.

Listener Registration Process (LREG)

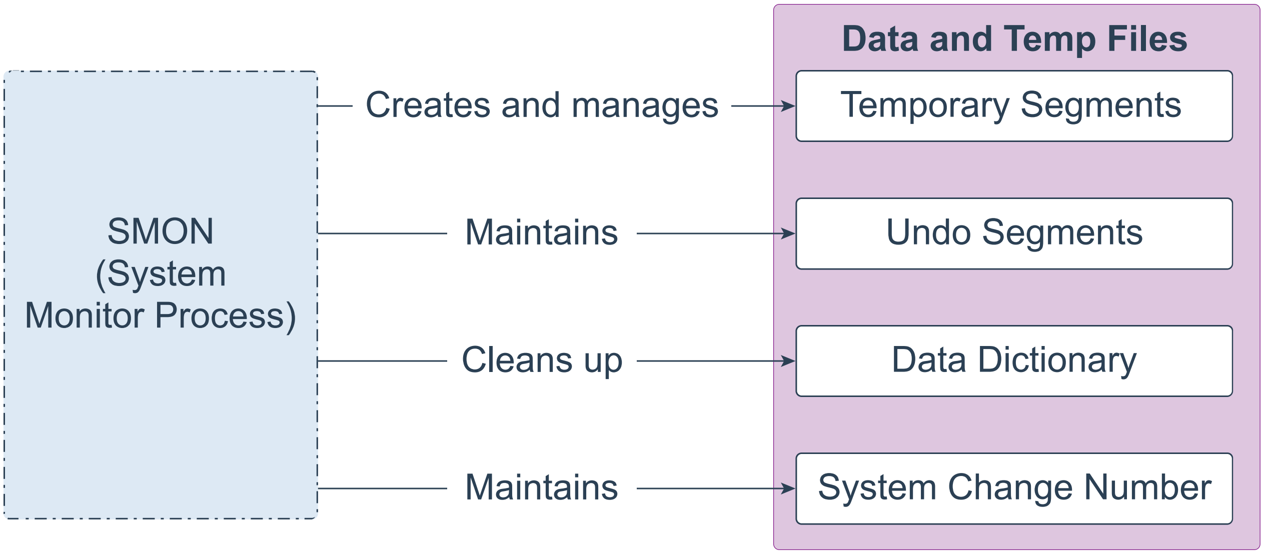


Listener Registration Process (LREG) is a background process that notifies the listeners about instances, services, handlers, and endpoints.

LREG can run as a thread or an operating system process. In addition to database instances, LREG also runs on Oracle Automatic Storage Management (ASM) instances and Oracle Real Application Clusters (RAC).

For a complete list of background processes, see Background Processes.

System Monitor Process (SMON)



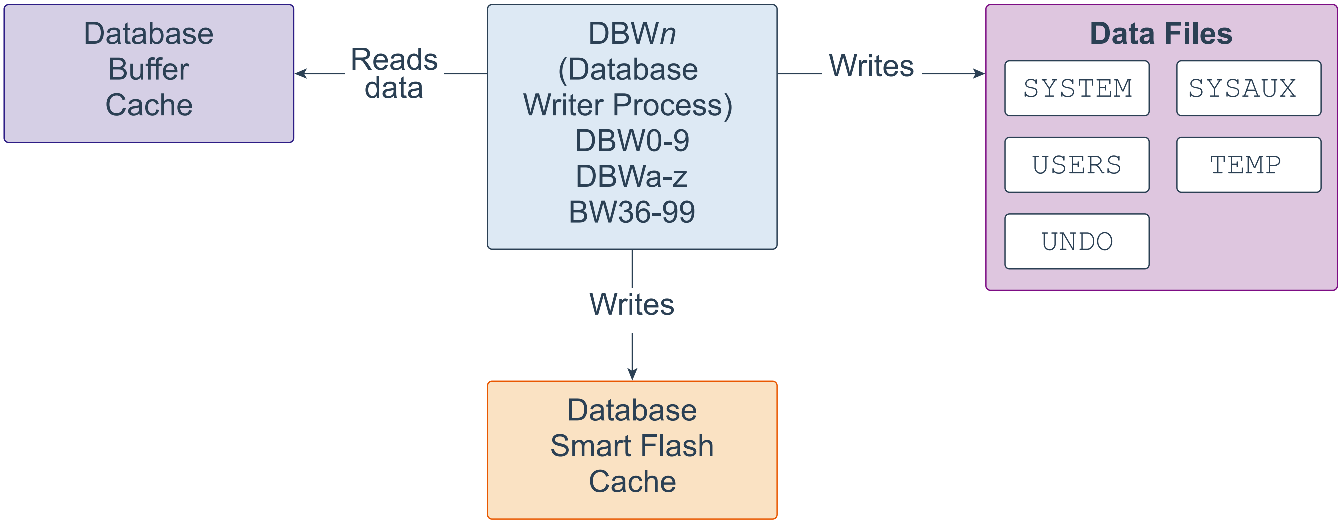
System Monitor Process (SMON) is a background process that performs many database maintenance tasks, including the following:

- Creates and manages the temporary tablespace metadata, and reclaims space used by orphaned temporary segments
- Maintains the undo tablespace by onlining, offlining, and shrinking the undo segments based on undo space usage statistics
- Cleans up the data dictionary when it is in a transient and inconsistent state
- Maintains the System Change Number (SCN) to time mapping table used to support Oracle Flashback features

SMON is resilient to internal and external errors raised during background activities. SMON can run as a thread or an operating system process. In an Oracle Real Application Clusters (RAC) database, the SMON process of one instance can perform instance recovery for other instances that have failed.

For a complete list of background processes, see Background Processes.

Database Writer Process (DBWn)


Database Buffer Cache
Reads data
DBW*n* (Database Writer Process) DBW0-9 DBWa-z BW36-99
Writes
Data Files
SYSTEM
SYSAUX
USERS
TEMP
UNDO
Writes
Database Smart Flash Cache

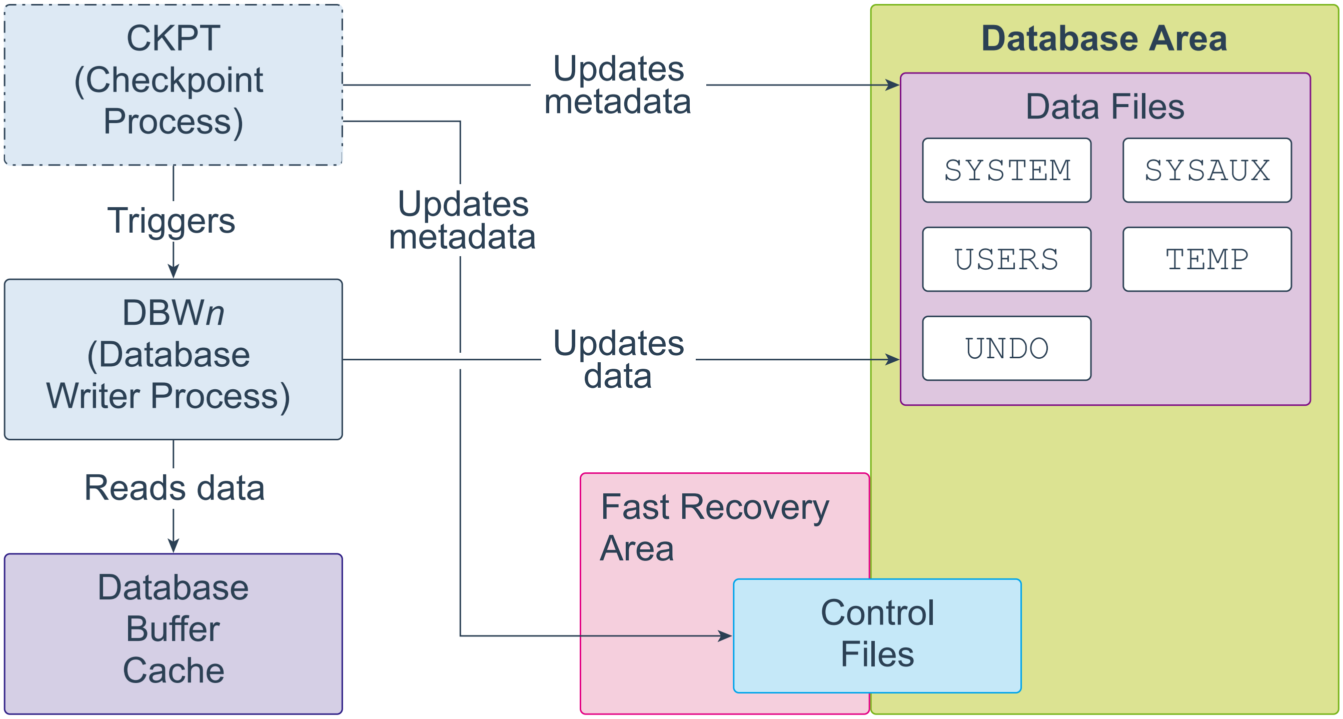
Database Writer Process (DBWn) is a background process that primarily writes data blocks to disk. It also handles checkpoints, file open synchronization, and logging of Block Written records. DBWn also writes to the Database Smart Flash Cache (Flash Cache), when Flash Cache is configured.

In many cases the blocks that DBWn writes are scattered throughout the disk. Thus, the writes tend to be slower than the sequential writes performed by the Log Writer Process (LGWR). DBWn performs multi-block writes when possible to improve efficiency. The number of blocks written in a multi-block write varies by operating system.

The `DB_WRITER_PROCESSES` initialization parameter specifies the number of Database Writer Processes. There can be 1 to 100 Database Writer Processes. The names of the first 36 Database Writer Processes are DBW0-DBW9 and DBWa-DBWz. The names of the 37th through 100th Database Writer Processes are BW36-BW99. The database selects an appropriate default setting for the `DB_WRITER_PROCESSES` parameter or adjusts a user-specified setting based on the number of CPUs and processor groups.

For a complete list of background processes, see Background Processes.

Checkpoint Process (CKPT)



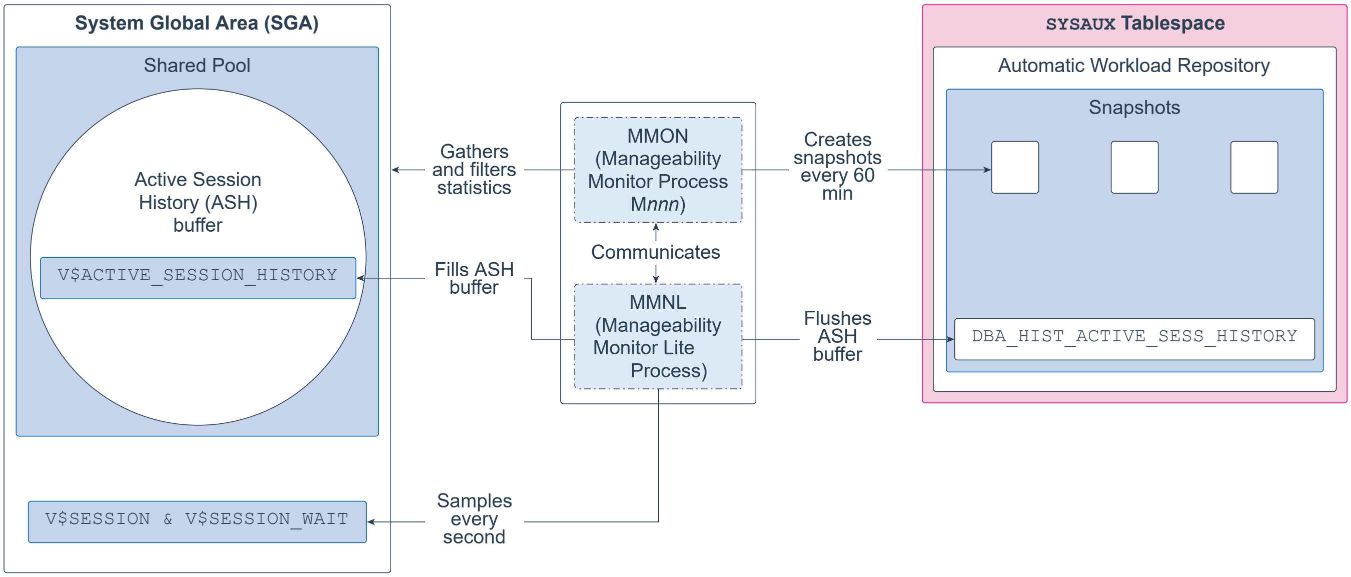
Checkpoint Process (CKPT) is a background process that, at specific times, starts a checkpoint request by messaging Database Writer Process (DBWn) to begin writing dirty buffers. On completion of individual checkpoint requests, CKPT updates data file headers and control files to record the most recent checkpoint.

CKPT checks every three seconds to see whether the amount of memory exceeds the value of the `PGA_AGGREGATE_LIMIT` initialization parameter, and if so, takes action.

CKPT can run as a thread or an operating system process. In addition to database instances, CKPT also runs on Oracle Automatic Storage Management (ASM) instances.

For a complete list of background processes, see Background Processes.

Manageability Monitor Process (MMON) and Manageability Monitor Lite Process (MMNL)


System Global Area (SGA)
Shared Pool
Active Session History (ASH) buffer
V$ACTIVE_SESSION_HISTORY
V$SESSION & V$SESSION_WAIT
Gathers and filters statistics
Fills ASH buffer
Samples every second
MMON (Manageability Monitor Process Mnnn)
Communicates
MMNL (Manageability Monitor Lite Process)
Creates snapshots every 60 min
Flushes ASH buffer
SYSAUX Tablespace
Automatic Workload Repository
Snapshots
DBA_HIST_ACTIVE_SESS_HISTORY

Manageability Monitor Process (MMON) and Manageability Monitor Lite Process (MMNL) are background processes that perform tasks related to the Automatic Workload Repository (AWR). The AWR is a repository of historical performance data that includes cumulative statistics for the system, sessions, individual SQL statements, segments, and services. It is used for problem detection and self-tuning purposes.

MMON gathers a variety of memory statistics from the SGA, filters them, and then creates snapshots of those statistics every 60 minutes in the Automatic Workload Repository (AWR). 60 minutes is the default value and can be altered. It also performs Automatic Database Diagnostic Monitor (ADDM) analysis and issues alerts for metrics that exceed their threshold values.

MMNL gathers session statistics (such as the user ID, state, the machine, and the SQL it is executing) and stores them in the Active Session History (ASH) buffer. Specifically, MMNL samples the `V$SESSION` and `V$SESSION_WAIT` views every second in the SGA and then records that data in the `V$ACTIVE_SESSION_HISTORY` view. Inactive sessions are not sampled. The ASH is designed as a rolling buffer in memory, and therefore, earlier information is overwritten when needed. When the ASH buffer becomes full or when MMON takes a snapshot, MMNL flushes (empties) the ASH buffer into the `DBA_HIST_ACTIVE_SESS_HISTORY` view in the AWR. Because space is expensive, only one in every 10 entries is flushed. MMNL also computes metrics.

Both MMON and MMNL can run as threads or as an operating system processes. In addition to database instances, MMON and MMNL also run on Automatic Storage Management (ASM) instances.

For more information, see:

- Managing the SYSAUX Tablespace
- Managing the Automatic Workload Repository
- Active Session History Statistics

For a complete list of background processes, see Background Processes.

Recoverer Process (RECO)

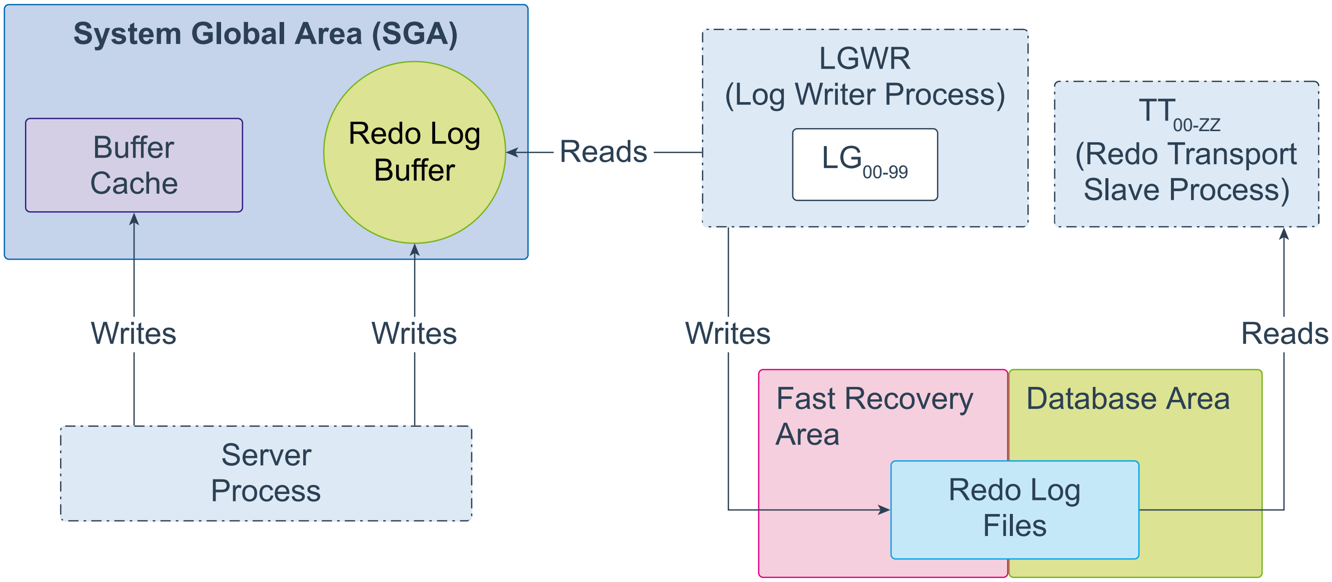
RECO
(Recoverer Process)

Recoverer Process (RECO) is a background process that resolves distributed transactions that are pending because of a network or system failure in a distributed database.

RECO can run as a thread or as an operating system process.

For a complete list of background processes, see Background Processes.

Log Writer Process (LGWR)



Log Writer Process (LGWR) is a background process that writes redo log entries sequentially into a redo log file. Redo log entries are generated in the redo log buffer of the System Global Area (SGA). If the database has a multiplexed redo log, then LGWR writes the same redo log entries to all of the members of a redo log file group.

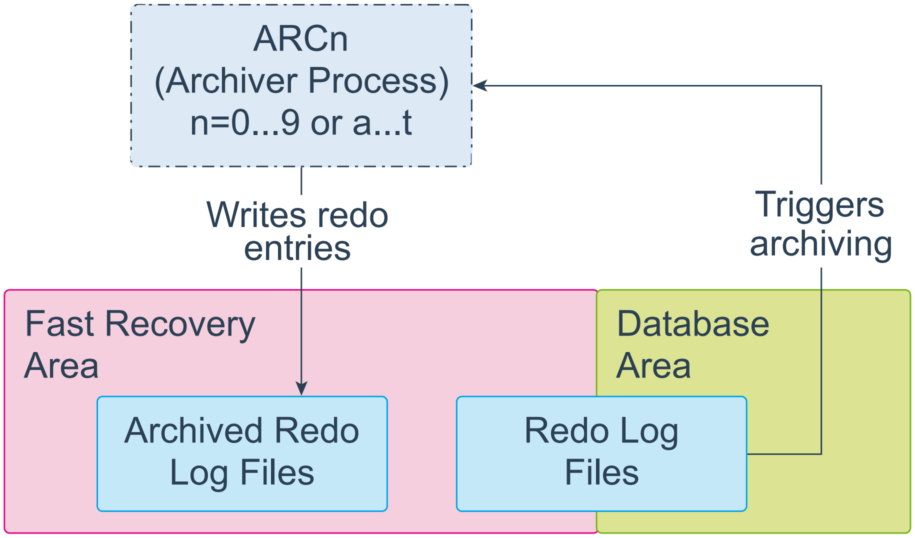
LGWR handles the operations that are very fast, or must be coordinated, and delegates operations to the Log Writer Worker helper processes (LGnn) that could benefit from concurrent operations, primarily writing the redo from the log buffer to the redo log file and posting the completed write to the foreground process that is waiting.

The Redo Transport Worker Process (TT00-zz) ships redo from the current online and standby redo logs to remote standby destinations configured for Asynchronous (ASYNC) redo transport.

LGWR can run as a thread or as an operating system process. In addition to database instances, LGWR also runs on Oracle ASM instances. Each database instance in an Oracle Real Application Clusters (RAC) configuration has its own set of redo log files.

For a complete list of background processes, see Background Processes.

Archiver Process (ARCn)



Archiver Processes (ARCn) are background processes that exist only when the database is in `ARCHIVELOG` mode and automatic archiving is enabled, in which case ARCn automatically archives online redo log files. Log Writer Process (LGWR) cannot reuse and overwrite an online redo log group until it has been archived.

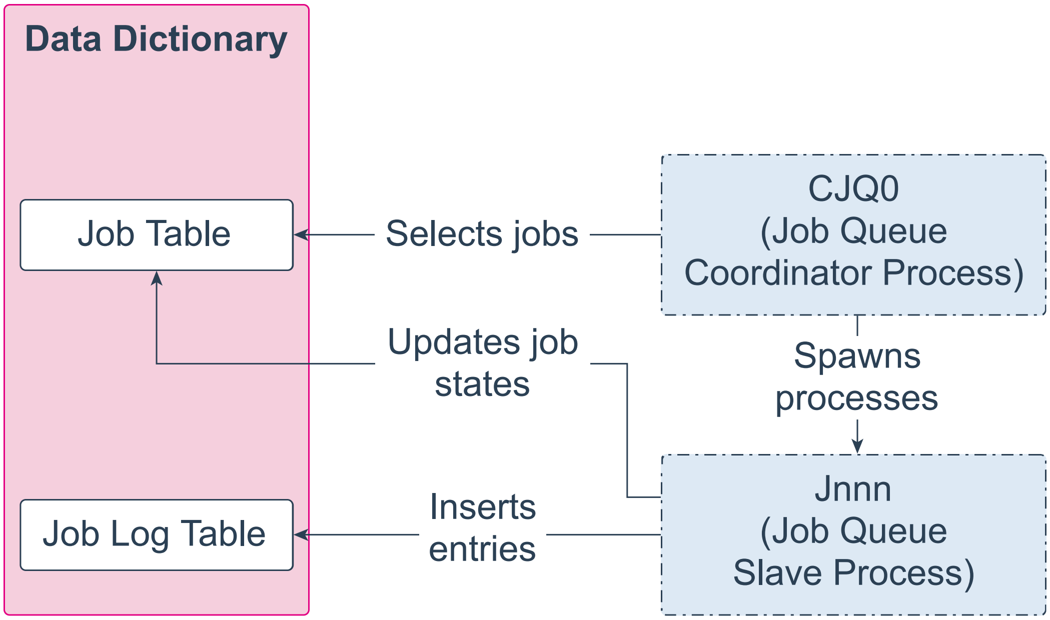
The database starts multiple archiver processes as needed to ensure that the archiving of filled online redo logs does not fall behind. Possible processes include ARC0-ARC9 and ARCa-ARCt (31 possible destinations).

The `LOG_ARCHIVE_MAX_PROCESSES` initialization parameter specifies the number of ARCn processes that the database initially invokes. If you anticipate a heavy workload for archiving, such as during bulk loading of data, you can increase the maximum number of archiver processes. There can also be multiple archive log destinations. It is recommended that there be at least one archiver process for each destination.

ARCn can run as a thread or as an operating system process.

For a complete list of background processes, see Background Processes.

Job Queue Coordinator Process (CJQ0)



Job Queue Coordinator Process (CJQ0) is a background process that selects jobs that need to be run from the data dictionary and spawns Job Queue Worker Processes (Jnnn) to run the jobs. CJQ0 is automatically started and stopped as needed by Oracle Scheduler. The `JOB_QUEUE_PROCESSES` initialization parameter specifies the maximum number of processes that can be created for the execution of jobs. CJQ0 starts only as many job queue processes as required by the number of jobs to run and available resources.

A Job Queue Worker Process (Jnnn) executes jobs assigned by the job coordinator. When a job is picked for processing, the job worker does the following:

- Gathers all the metadata needed to run the job, for example, program arguments and privilege information.
- Starts a database session as the owner of the job, starts a transaction, and then starts executing the job.
- Once the job is complete, the worker commits and ends the transaction.
- Closes the session.

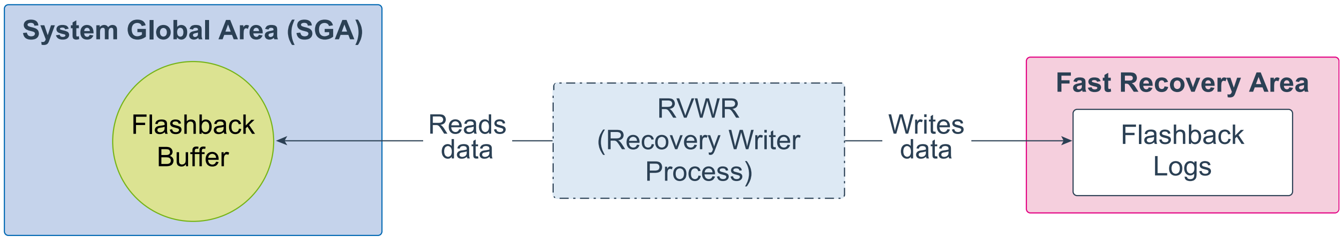
When a job is done, the workers do the following:

- Reschedule the job if required
- Update the state in the job table to reflect whether the job has completed or is scheduled to run again
- Insert an entry into the job log table
- Update the run count, and if necessary, failure and retry counts
- Clean up
- Look for new work (if none, they go to sleep)

Both CJQ0 and Jnnn can run as threads or as operating system processes.

For a complete list of background processes, see Background Processes.

Recovery Writer Process (RVWR)

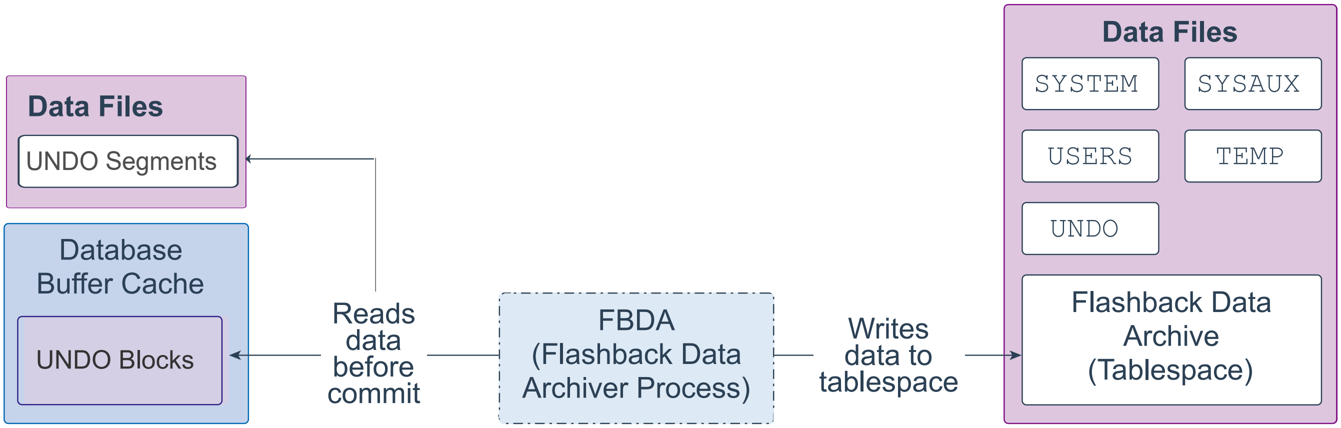


Recovery Writer Process (RVWR) is a background process that is used to flashback an entire database or a pluggable database. That is, it undoes transactions from the current state of the database to a time in the past, provided you have the required flashback logs. When flashback is enabled or when there are guaranteed restore points, RVWR writes flashback data to flashback database logs in the fast recovery area.

RVWR can run as a thread or as an operating system process.

For a complete list of background processes, see Background Processes.

Flashback Data Archiver Process (FBDA)



Flashback Data Archiver Process (FBDA) is a background process that provides the ability to track and store transactional changes to a table over its lifetime. This way, you can flashback tables back in time to restore the way they were.

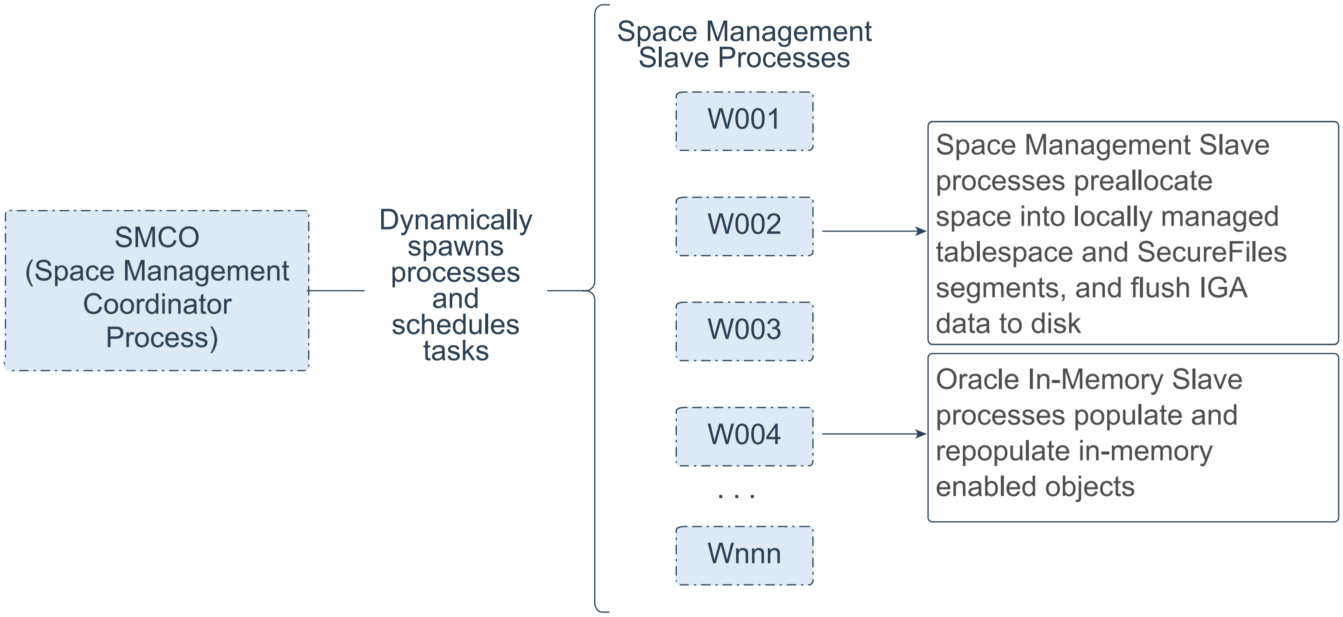
When a transaction that modifies a tracked table commits, FBDA checks for new undo being generated, filters what is relevant to objects marked for archival and copies that undo information into the Flashback Data Archive tablespace. FBDA maintains metadata on the current rows and tracks how much data has been archived.

FBDA is also responsible for automatically managing the flashback data archive for space, organization (partitioning tablespaces), and retention. FBDA also keeps track of how far the archiving of tracked transactions has progressed.

FBDA can run as a thread or as an operating system process.

For a complete list of background processes, see Background Processes.

Space Management Coordinator Process (SMCO)


Space Management Slave Processes
SMCO (Space Management Coordinator Process)
Dynamically spawns processes and schedules tasks
W001
W002
W003
W004
. . .
Wnnn
Space Management Slave processes preallocate space into locally managed tablespace and SecureFiles segments, and flush IGA data to disk
Oracle In-Memory Slave processes populate and repopulate in-memory enabled objects

Space Management Coordinator Process (SMCO) is a background process that schedules the execution of various space management tasks, including proactive space allocation and space reclamation. SMCO dynamically spawns Space Management Worker Processes (Wnnn) to implement these tasks. Note that the In-Memory Coordinator Process (IMCO) is a background process that initiates background population and repopulation of columnar data.

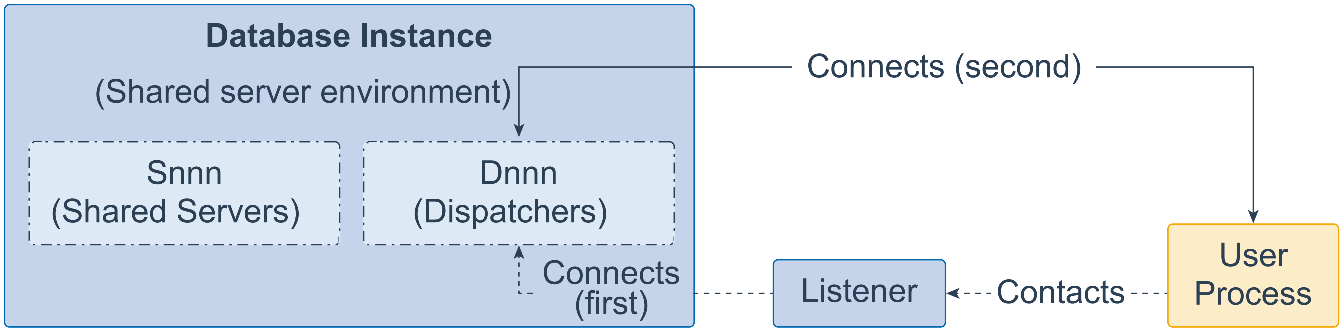
W*nnn* worker processes perform work on behalf of Space Management and on behalf of the Oracle In-Memory option.

- Wnnn processes are worker processes dynamically spawned by SMCO to perform space management tasks in the background. These tasks include preallocating space into locally managed tablespace and SecureFiles segments based on space usage growth analysis, and reclaiming space from dropped segments. The tasks also include fast ingest deferred inserts. After being started, the worker acts as an autonomous agent. After it finishes task execution, it automatically picks up another task from the queue. The process terminates itself after being idle for a long time.
- W*nnn* processes populate and repopulate in-memory enabled objects. The In-Memory Coordinator Process (IMCO) initiates background population and repopulation of columnar data. The IMCO background process and foreground processes will utilize Wnnn workers for population and repopulation. W*nnn* processes are utilized by IMCO for prepopulation of in-memory enabled objects with priority `LOW/MEDIUM/HIGH/CRITICAL`, and for repopulation of in-memory objects. In-memory populate and repopulate tasks running on Wnnn workers are also initiated from foreground processes in response to queries and DMLs that reference in-memory enabled objects.

Both SMCO and W*nnn* can run as threads or as operating system processes.

For a complete list of background processes, see Background Processes.

Dispatcher Process (Dnnn) and Shared Server Process (Snnn)



In a shared server architecture, clients connect to a Dispatcher Process (Dnnn), which creates a virtual circuit for each connection. When the client sends data to the server, the dispatcher receives the data into the virtual circuit and places the active circuit on the common queue to be picked up by an idle Shared Server process (Snnn). The Snnn then reads the data from the virtual circuit and performs the database work necessary to complete the request. When the Snnn must send data to the client, the Snnn writes the data back into the virtual circuit and the Dnnn sends the data to the client. After the Snnn completes the client request, it releases the virtual circuit back to the Dnnn and is free to handle other clients.

Both Snnn and Dnnn can run as threads or as operating system processes. In addition to database instances, Dnnn also runs on shared servers.

For information about how Dnnn and Snnn interact with the Large Pool, see the Large Pool slide. For a complete list of background processes, see Background Processes.